# iRMX® Programming Concepts for DOS and Windows

Order Number: 469154-003

In the United States, additional copies of this manual or other Intel literature may be obtained by writing:

Literature Distribution Center
Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

Or you can call the following toll-free number: 1-800-548-4725

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are printed on the last page of this document. Contact your local sales office to obtain the latest specifications before placing your order.

Intel Corporation (Intel) makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel assumes no responsibility for any errors that may appear in this document. Intel makes no commitment to update nor to keep current the information contained in this document. No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel. Intel retains the right to make changes to these specifications at any time, without notice.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License Agreement.
**U.S. GOVERNMENT RESTRICTED RIGHTS:** These software products and documentation were developed at private expense and are provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR 52.227-14 and DFAR 252.227-7013 et seq. or its successor.

The Intel logo, i960, Pentium, and iRMX are registered trademarks of Intel Corporation, registered in the United States of America and other countries. Above, i287, i386, i387, i486, Intel287, Intel386, Intel387, Intel486, Intel487 and EtherExpress are trademarks of Intel Corporation.

Adaptec is a registered trademark of Adaptec, Inc. AT, IBM and PS/2 are registered trademarks and PC/XT is a trademark of International Business Machines Corporation. All Borland products are trademarks or registered trademarks of Borland International, Inc. CodeView, Microsoft, MS, MS-DOS and XENIX are registered trademarks of Microsoft Corporation. Comtrol is a registered trademark and HOSTESS is a trademark of Comtrol Corporation. DT2806 is a trademark of Data Translation, Inc. Ethernet is a registered trademark of Xerox Corporation. Hayes is a registered trademark of Hayes Microcomputer Products. Hazeltine and Executive 80 are trademarks of Hazeltine Corporation. Hewlett-Packard is a registered trademark of Hewlett-Packard Co. Maxtor is a registered trademark of Maxtor Corporation. MIX® is a registered trademark of MIX Software, Incorporated. MIX is an acronym for Modular Interface eXtension. MPI is a trademark of Centralp Automatismes (S.A.). NetWare and Novell are registered trademarks of Novell Corp. NFS is a trademark of Sun Microsystems, Inc. Phar Lap is a trademark of Phar Lap Software, Inc. Soft-Scope is a registered trademark of Concurrent Sciences, inc. TeleVideo is a trademark of TeleVideo Systems, Inc. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited. VAX is a registered trademark and VMS is a trademark of Digital Equipment Corporation. Visual Basic and Visual C++ are trademarks of Microsoft Corporation. All Watcom products are trademarks or registered trademarks of Watcom International Corp. Windows, Windows 95 and Windows for Workgroups are registered trademarks and Windows NT is a trademark of Microsoft in the U.S. and other countries. Wyse is a registered trademark of Wyse Technology. Zentec is a trademark of Zentec Corporation. Other trademarks and brands are the property of their respective owners.

Copyright © 1991, 1993 and 1995 Intel Corporation, All Rights Reserved

| **REVISION HISTORY** | | **DATE** |
|---|---|---|
| -001 | Original Issue | 12/91 |
| -002 | Revision One | 11/93 |
| -003 | Revision Two. Added new DDE information and re-titled manual from *iRMX for Windows Programming Concepts* | 11/95 |

# Quick Contents

Chapter 1. Introduction

Chapter 2. DOS Real-time Extension

Chapter 3. VM86 Protected Mode Extensions

Chapter 4. Making DOS and ROM BIOS System Calls

Chapter 5. General Information

Chapter 6. Remote Launch

Chapter 7. Using DDE

Appendix A. Default Configuration

Index

Service Information

# Notational Conventions

Most of the references to system calls in the text and graphics use C syntax instead of PL/M (for example, the system call **send_message** instead of **send$message**). If you are working in C, you must use the C header files, *rmx_c.h*, *udi_c.h* and *rmx_err.h*. If you are working in PL/M, you must use dollar signs ($) and use the *rmxplm.ext* and *error.lit* header files.

This manual uses the following conventions:

- Syntax strings, data types, and data structures are provided for PL/M and C respectively.
- All numbers are decimal unless otherwise stated. Hexadecimal numbers include the `H` radix character (for example, `0FFH`). Binary numbers include the `B` radix character (for example, `11011000B`).
- Bit 0 is the low-order bit. If a bit is set to 1, the associated description is true unless otherwise stated.
- `Data structures and syntax strings appear in this font.`
- **System call names and command names appear in this font.**
- PL/M data types such as BYTE and SELECTOR, and iRMX data types such as STRING and SOCKET are capitalized. All C data types are lower case except those that represent data structures.
- The following OS layer abbreviations are used. The Nucleus layer is unabbreviated.

| | |
|---|---|
| AL | Application Loader |
| BIOS | Basic I/O System |
| EIOS | Extended I/O System |
| HI | Human Interface |
| UDI | Universal Development Interface |

- Whenever this manual describes I/O operations, it assumes that tasks use BIOS calls (such as **rq_a_read**, **rq_a_write**, and **rq_a_special**). Although not mentioned, tasks can also use the equivalent EIOS calls (such as **rq_s_read**, **rq_s_write**, and **rq_s_special**) or UDI calls (**dq_read** or **dq_write**) to do the same operations.

# Contents

## 1 Introduction

Understanding the Environments ... 12
Running DOS and the iRMX® OS on the Same System ... 13
VM86 Dispatcher ... 14
VM86 Protected Mode Extensions ... 14
Real-time Extension ... 15
Windows Support ... 16
Making DOS/ROM BIOS System Calls from the iRMX OS ... 17
File Access ... 18
EDOS File Driver ... 20
Networking ... 21
File and Device Drivers ... 22
Loadable File and Device Drivers ... 22
System Configuration ... 22

## 2 DOS Real-Time Extension

RTE System Calls ... 23
RQEGetRmxStatus Call ... 25
DLL for RTE Interfaces ... 25
RTE Files ... 26
RTE Objects Limitation ... 26
Making an RTE System Call ... 27
Using RTE Functions ... 28
DOS RTE Demonstration ... 28
Example: Running the Demonstration Program ... 30
Mailboxes (Objects) Functions ... 31
Mailboxes (Data) Functions ... 32
Semaphore Functions ... 33
PVAM Segment Functions ... 34
Descriptor Functions ... 35
Data Transfer Functions ... 36
Example: WTERM Demonstration Program ... 36

## 3 VM86 Protected Mode Extensions

Installing VM86 Protected Mode Extensions ............ 37
iRMX Interrupt Levels ............ 38
Extension Procedure Operation: DOS Interrupt Handling ............ 39
Deletion Handler Operation ............ 40
Extension System Call Restrictions ............ 40
Extension Installation Examples ............ 41
Installing an Extension from the iRMX Operating System ............ 41
Initiating an Extension from DOS ............ 41

## 4 Making DOS and ROM BIOS System Calls

Making DOS and ROM BIOS Calls from an iRMX Application ............ 43
Example: Get Free Disk Space ............ 44
Get Redirection List Entry Example ............ 46
Setting the DOS Data Structure ............ 46

## 5 General Information

Interrupt Virtualization and Determinism ............ 49
Real-time Fence ............ 50
iRMX-NET Access From a DOS Server ............ 50

## 6 Remote Launch

Invoking iRMX Programs from DOS or Windows ............ 51
Using the iRMX Load Server with iRMX for Windows ............ 53
Using MS-NET with the PCLINK2 ............ 54
Using RLS.JOB ............ 55
Running RLS.JOB ............ 55
Using DLC.EXE ............ 56
Using WLC.EXE ............ 56
Creating the WLC Icon ............ 56
Running WLC.EXE ............ 57

## 7 Using DDE


Real-time DDE Capabilities .......... 59
  DDE Terminology .......... 60
  Communications Protocols .......... 61
  iRMX DDE Capabilities .......... 64
  iRMX DDE Restrictions .......... 64
    DDE Router Internal Configuration Limits .......... 65
Configuring the Windows DDE Routers .......... 65
  Enabling DDE Routing on the Windows System .......... 66
Preparing the Windows Environment for NetDDE .......... 67
Preparing the iRMX Environment for NetDDE .......... 68
DDE Programming .......... 69
  DDE Messages .......... 69
  iRMX Client to Windows Server Conversations .......... 70
    Initiating Conversations .......... 72
    Transferring Single Items .......... 73
    Submitting Commands to the Server .......... 74
    Using Warm Links .......... 77
  Windows Client to iRMX Server Conversations .......... 79
    Establishing Conversations .......... 81
    Responding to Data Item Requests .......... 82
    Responding to Hot Links .......... 83
    Handling Termination Requests .......... 85
DDE Library .......... 86
  Summary of DDE Library Functions .......... 86
  Error Codes .......... 88
  DDE Library Functions .......... 88
client_dde_close_link .......... 89
client_dde_execute .......... 90
client_dde_initiate .......... 91
client_dde_open_hot_link .......... 93
client_dde_open_warm_link .......... 94
client_dde_poke .......... 95
client_dde_request .......... 96
client_dde_run_application .......... 97
client_dde_terminate .......... 98
client_link_callback .......... 99
dde_library_init .......... 100
server_conversation_callback .......... 101
server_data_callback .......... 102
server_dde_register .......... 104
server_dde_terminate .......... 105

server_dde_update_link ........ 106
Establishing DDE Communications Between Windows
and iRMX Applications ........ 107
DC Motor Example ........ 107
DC Motor Description ........ 109
DDE Client Supervisor / DDE Server Controller Implementation ........ 110
Visual Basic Supervisor Implemented as DDE Client ........ 110
iRMX Controller Implemented as a DDE Server ........ 110
Running the Example ........ 111
DDE Server Supervisor/DDE Client Controller Implementation ........ 112
Visual Basic Supervisor Implemented as a DDE Server ........ 112
iRMX Controller Implemented as a DDE Client ........ 112
Running the Example ........ 113

## A iRMX for Windows Default Configuration

Sub-System Configuration ........ 116
Memory Configuration ........ 116
Human Interface Configuration ........ 117
Application Loader Configuration ........ 117
Extended I/O System Configuration ........ 118
Basic I/O System Configuration ........ 118
Device Drivers Configuration ........ 119
System Debug Monitor Configuration ........ 120
Nucleus Configuration ........ 120
Nucleus Communication Service Configuration ........ 121
VM86 Dispatcher Reserved Interrupts Configuration ........ 121

## Index

123

## Service Information

Inside Back Cover

## Tables


1-1. Facilities for Supporting Various iRMX OS and Windows Configurations .......... 13
2-1. RTE System Calls .......... 24
3-1. iRMX Interrupt Levels .......... 38
7-1. DDE Library Client Functions .......... 71
7-2. DDE Library Server Functions .......... 80
7-3. DDE Library Summary .......... 86
7-4. Error Codes .......... 88
A-1. Sub-Systems Options .......... 116
A-2. Memory Options .......... 116
A-3. Human Interface Options .......... 117
A-4. Application Loader Options .......... 117
A-5. EIOS Options .......... 118
A-6. BIOS Options .......... 118
A-7. Device Drivers Options .......... 119
A-8. System Debug Monitor Options .......... 120
A-9. Nucleus Options .......... 120
A-10. Nucleus Communication Service Options .......... 121
A-11. DOS Extender Reserved Interrupts .......... 121


## Figures


1-1. Making Nucleus System Calls with the DOS Real-time Extension .......... 15
1-2. Making DOS and ROM BIOS Requests from an iRMX Application .......... 17
1-3. Using Networking to Access Files on the iRMX File System .......... 19
1-4. Accessing DOS Files with the EDOS File Driver .......... 20
6-1. Launching Commands from Multiple Clients to a Single Server .......... 52
6-2. Launching Commands within a Single System Client and Server .......... 52
6-3. DOS-only System Accessing an iRMX System .......... 54
7-1. RTE Communication between iRMX and Windows within the Same System .......... 62
7-2. NetBIOS Communication between iRMX and Windows within the Same System .......... 62
7-3. NetBIOS Communication between iRMX and Windows on Different Systems .......... 63
7-4. TCP/IP Communication between iRMX and Windows on Different Systems .......... 63
7-5. iRMX to Windows DDE Conversations .......... 70
7-6. Windows to iRMX Conversations .......... 79
7-7. Supervisor/Controller Communications .......... 108
7-8. The Visual Basic Supervisor .......... 109

# Introduction 1

This manual discusses the programming concepts necessary to produce real-time applications for an environment that includes DOS, Microsoft Windows and the iRMX® OS. While its primary focus is on the iRMX for Windows OS (which runs concurrently with DOS/Windows on the same system), certain items in this manual apply to networked systems running DOS/Windows and any iRMX OS.

> **Note**
> The iRMX for Windows OS is supported only for Windows 3.1 and Windows 3.11.

This manual is for programmers who are familiar with:

- Applications programming in the DOS and Windows environment
- Terms and concepts for the iRMX OS

  See also: Introducing the iRMX Operating Systems, System Concepts

- C or PL/M programming language

  See also: *iC-386 Compiler User's Guide, PL/M-386 Programmer's Guide*

The iRMX for Windows OS provides a set of powerful extensions to DOS and Windows. With it you can develop DOS and Windows applications that incorporate the preemptive, priority-based multitasking and real-time response of the iRMX OS.

iRMX for Windows enables:

- MS-DOS or PC-DOS OSs to run concurrently with the iRMX OS on the same microprocessor and to share the same console
- Existing DOS Real Mode application programs, including most off-the-shelf applications, to run under DOS with no modification

- Standard and Real Mode Windows applications to run under DOS
- Existing iRMX application programs to run under iRMX for Windows with no modification, while maintaining real-time performance
- DOS application programs to make iRMX Nucleus system calls and to communicate directly with 32-bit Protected Mode iRMX application programs
- iRMX application programs to make DOS and ROM BIOS system calls from within an iRMX task
- DOS programs to access iRMX files and iRMX programs to access DOS files
- Preconfigured and loadable file and device drivers and system jobs
- Simultaneous access to network services from DOS and the iRMX OS
- DOS applications to access expanded memory using an Intel Above™ Board or an equivalent board

## Understanding the Environments

Some of the facilities discussed in this manual can be used in two environments. One environment consists of iRMX for Windows and DOS/Windows within a single system. The other environment consists of networked systems where one system can be any iRMX OS communicating with a system running DOS/Windows. These facilities include:

- Network Dynamic Data Exchange (DDE). You can use DDE communications based upon either the TCP/IP or OpenNET networking protocols to communicate between iRMX and Windows applications running on separate systems. Similarly, you can use DDE communications based upon either the OpenNET networking protocol or the DOS Real-time Extension (RTE) facility to communicate between iRMX and Windows applications running on the same system.
- Remote File Access. Using the NetBIOS interface to the OpenNET networking protocol and a standard DOS Network Redirector, DOS and Windows applications can access the iRMX file systems on a local iRMX for Windows system or remote systems running any iRMX OS.
- Remote Launch Facility. Using the NetBIOS interface to the OpenNET networking protocol, both the DOS Load Client (DLC) and the Windows Load Client (WLC) allow the launching of iRMX programs on either local or remote iRMX for Windows systems.

**Note**

When running DOS/Windows on a stand-alone system, the PCLINK2 Networking Adapter facilitates access to the OpenNET networking protocol. This access is restricted to systems running only Windows 3.1 or 3.11. The access is based upon a NetBIOS interface, which is built into Windows for Workgroups, Windows 95, and Windows NT.

Table 1-1 shows the facilities required to perform certain operations between the iRMX OS and Windows.

**Table 1-1. Facilities for Supporting Various iRMX OS and Windows Configurations**

| | **Windows PC, iRMX System, TCP/IP network** | **Windows PC, iRMX System, ISO network** | **iRMX for Windows, one PC** |
|---|---|---|---|
| **File Access** | NFS*, FTP* | NetBIOS (iRMX-NET) | NetBIOS (iRMX-NET) |
| **Remote Command Execution** | RCP*, RSH* | Windows Load Client (RLS) | Windows Load Client (RLS) |
| **Network DDE Facility** | TCP/IP* | NetBIOS | NetBIOS, RTE |
| **Virtual Terminal Support** | Telnet*, Rlogin* | Windows Load Client (RLS) | Windows Load Client (RLS), Wterm |

*See also: *TCP/IP and NFS for the iRMX Operating System*

# Running DOS and the iRMX® OS on the Same System

By itself, DOS does not use the advanced features of the Intel386™, Intel486™ and Pentium® microprocessors such as Protected Mode Addressing.

The iRMX OS exploits more of the features of these microprocessors, allowing tasks to run concurrently and to reside in up to 4 Gbytes of memory. Under the iRMX OS, these microprocessors also run in 32-bit Protected Mode.

iRMX for Windows encapsulates DOS as an iRMX task and runs that task in Virtual 86 Mode (VM86). The encapsulated DOS task (DOS and its application programs) runs in the first 1 Mbyte of memory, and the iRMX OS runs in the remaining memory.

A terminate and stay resident (TSR) program, *rmxtsr.exe*, provides a small buffer in DOS memory which enables DOS and the iRMX OS to exchange data. Two loadable jobs, *himem.job* and *smw.job*, provide DOS access to extended memory and allow Standard Mode Windows to run. These jobs are located in the *\rmx386\jobs* directory.

See also: Loadable system jobs, *System Configuration and Administration*, Windows Support, in this chapter

## VM86 Dispatcher

The VM86 Dispatcher enables DOS to run as a task under iRMX for Windows by:

- Switching the microprocessor addressing mode, depending on which OS is running
- Ensuring that interrupts are handled by the appropriate OS
- Providing file sharing between the OSs
- Preventing hardware resource conflicts between the OSs

The VM86 Dispatcher is preconfigured into iRMX for Windows as an iRMX first-level system job.

See also: System jobs, *System Concepts*

DOS is not supplied with iRMX for Windows; you must install PC-DOS or MS-DOS OS before you can run iRMX for Windows. You can install compatible off-the-shelf DOS applications before you run iRMX for Windows.

## VM86 Protected Mode Extensions

The VM86 Protected Mode extensions allow you to use Protected Mode services provided by the iRMX OS. Using these extensions, DOS or Windows application programs running in VM86 mode can access Protected Mode services such as 4 Gbyte addressing, as well as the iRMX system calls.

The Intel-supplied VM86 Protected Mode extensions provided by the VM86 Dispatcher include:

- DOS Real-time extension (RTE)
- Network Redirector (NETRDR)

You can also write your own VM86 Protected Mode extensions.

See also: **rqe_set_vm86_extension**, in this manual and *System Call Reference*

## Real-time Extension

The DOS Real-time Extension (RTE) enables you to call some of the iRMX Nucleus system calls from within a DOS application program. By using the RTE, a DOS application program can communicate with a concurrently-running iRMX application program using standard iRMX techniques. The RTE includes system calls that create and delete iRMX objects and descriptors, read and write segments, and catalog and look up objects.

Figure 1-1 illustrates how a DOS application makes a Nucleus system call.

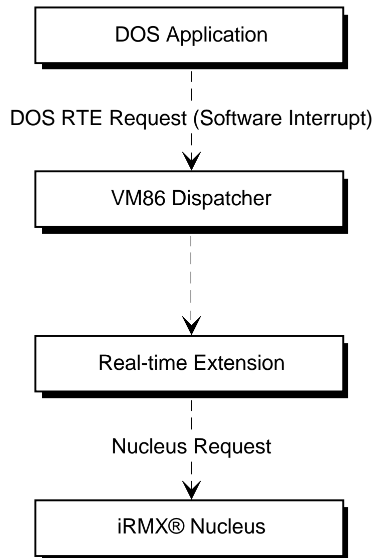



**Figure 1-1. Making Nucleus System Calls with the DOS Real-time Extension**

For example, the DOS application program may send or receive messages or data using a mailbox created by an iRMX application program. Similarly, an iRMX program may send or receive messages using a mailbox created by the DOS application program.

See also: DOS RTE, in this manual,
Windows- and DOS-specific system calls, *System Call Reference*

# Windows Support

iRMX for Windows supports the graphical environment of Windows. Windows can be used as an operator interface for real-time iRMX tasks. Use the DOS RTE to call iRMX Nucleus system calls from a Windows application.

Windows provides a powerful interface and development facility for iRMX real-time applications.

Standard Mode Windows applications are supported. These loadable system jobs provide this support:

- *himem.job* provides DOS extended memory and HMA services required by Standard Mode Windows.
- *smw.job* provides support for Standard Mode Windows by encapsulating the DOSX subsystem of Standard Mode Windows and running it in the context of the DOS task.

See also: Loadable Jobs, *System Configuration and Administration*

Applications using the iRMX DDE Library functions can pass data between Windows applications and real-time iRMX tasks. iRMX applications participate in Windows DDE communications using the iRMX OS's network communications facility. Therefore, Windows DDE applications can communicate over a network with these iRMX applications:

- iRMX for Windows applications on the same system
- Applications on any networked iRMX OS
- Applications on a different board in a Multibus II system

iRMX applications that use the DOS RTE for Windows DDE communications are restricted to communications within the local machine.

iRMX applications act as DDE clients or servers depending upon the application.

## Making DOS/ROM BIOS System Calls from the iRMX OS

iRMX for Windows enables iRMX programs to use the DOS and ROM BIOS software interrupt services, including any special ROM BIOS functions provided by add-in adapters, rather than implementing the same function as iRMX system calls.

Figure 1-2 illustrates how iRMX programs can make a DOS/ROM BIOS call.

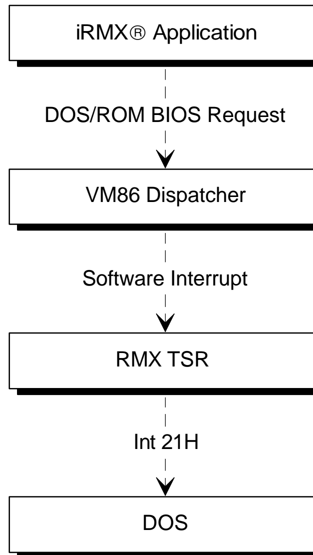



**Figure 1-2. Making DOS and ROM BIOS Requests from an iRMX Application**

For example, your iRMX application program can use the DOS **Get Free Disk Space** example to check available space on a network disk drive.

See also: Get Free Disk Space Example, Chapter 4,
**rqe_dos_request**, *System Call Reference*

# File Access

The DOS and iRMX file systems are inherently different. However, a file driver provided with the iRMX OS allows DOS and iRMX application programs to share files.

The Encapsulated DOS (EDOS) file driver enables iRMX application programs to access files on a DOS partition, storage device, or network drive that has a drive letter mapped to it. EDOS allows both DOS and iRMX partitions to store both DOS and iRMX files.

Applications on a DOS partition can also access files on an iRMX partition by using a network job and these configuration files:

- The PCNET NetBIOS driver, *pcnet.exe* (installed on the DOS partition).
- The network redirector job, *netrdr.job* (loaded on the iRMX partition).
- The iNA 960 networking job, *i*.job*, appropriate to your OS and the iRMX-NET client and server jobs (*remotefd.job* and *rnetserv.job*).

  See also: Using iRMX-NET in a DOS Environment, *System Configuration and Administration*

Under iRMX for Windows, you can choose whether to use only the DOS file system, or a partitioned file system including a DOS partition and one or more iRMX partitions. At least one DOS drive is required in an iRMX for Windows system.

See also: File access, *Command Reference*,
Installing iRMX for Windows, *Installation and Startup*

Figure 1-3 illustrates how DOS file requests are carried out by the I/O System.

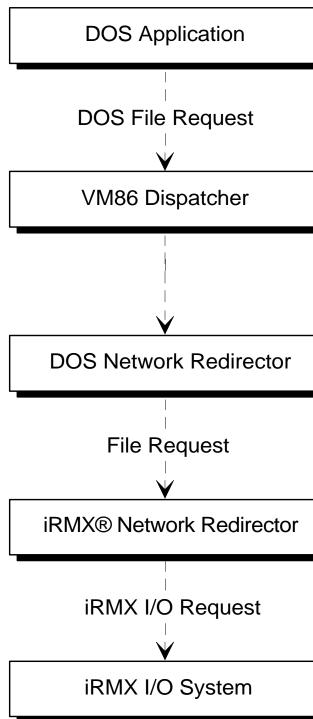



**Figure 1-3. Using Networking to Access Files on the iRMX File System**

The iRMX file system could be a separate drive or an iRMX partition on a drive containing both DOS and iRMX partitions, or even a remote drive accessed through iRMX-NET.

See also: File types, *System Concepts*

## EDOS File Driver

The EDOS file driver uses DOS as a file server to access DOS files. It maps iRMX file driver interfaces to DOS system calls. Therefore, files on the DOS drives appear to the iRMX application just as they would on an iRMX drive.

See also: **attachdevice** command, *Command Reference*

Figure 1-4 illustrates how iRMX applications make DOS file requests.

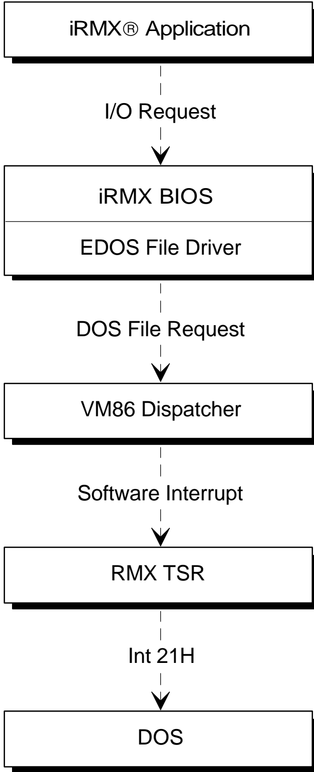



**Figure 1-4. Accessing DOS Files with the EDOS File Driver**

## Networking

iRMX for Windows supports both DOS and iRMX networking. This support provides a variety of capabilities:

- DOS and iRMX applications that communicate on the network run unchanged when they run within the same system.
- DOS files can be accessed from a remote file consumer without a dedicated file server.
- DOS and iRMX OSs running within the same system can share a single network controller.
- OpenNET networking support provides connections to computers running the DOS and UNIX OSs.
- DOS networking applications can use the network controller in a Multibus I or Multibus II system.
- iRMX applications can be remotely invoked from a DOS or Windows environment.

A PC running iRMX for Windows can run this network software:

- MS-Net client or server
- IBM PC LAN client or server
- Novell NetWare client
- Combinations of MS-Net and NetWare on one computer

PCs running iRMX for Windows can also run this iRMX network software:

- iRMX-NET consumer and server for remote file access, which can coexist with DOS network software
- iNA 960 jobs for a programmatic interface, but which cannot coexist with DOS network software.
- Null data link network jobs that allow DOS to access the iRMX file system without an Network Interface Controller (NIC).

See also: Network jobs, *i*.job*, *System Configuration and Administration*, Introduction, *Network User's Guide and Reference*

# File and Device Drivers

The iRMX for Windows software includes preconfigured file drivers and device drivers that can be loaded dynamically.

## Loadable File and Device Drivers

These driver allow you to write procedures to invoke and interface to additional custom, random access, and terminal hardware.

See also: Loadable file and device drivers, *Driver Programming Concepts* and *System Configuration and Administration*

# System Configuration

iRMX for Windows is preconfigured to run in the DOS and Windows environment, however you may change some aspects of the OS for a particular application.

Certain parts of the OS are loadable, including loadable file and device drivers and loadable jobs.

You load these elements into the system with the **sysload** command in the *:config:loadinfo* file. Loadable device drivers allow you to write procedures to invoke and interface to additional custom, random access, and terminal hardware. Loadable file drivers enable you to include custom file drivers.

The OS includes an *rmx.ini* file for load-time configuration. As layers of the OS boot, they read entries from this file. The *rmx.ini* file contains entries that match settings preconfigured into iRMX for Windows. You can modify the existing entries to fine-tune your use of the OS.

You can also use the Interactive Configuration Utility (ICU) to change the configuration of the iRMX for Windows or iRMX for PCs OSs.

See also: Loadable jobs and drivers, *System Configuration and Administration*,
Loadable device drivers, *Driver Programming Concepts*,
Physical device names, *Command Reference,*
ICU Quick Reference, *ICU User's Guide and Quick Reference*

# DOS Real-Time Extension 2

The DOS Real-time Extension (RTE) enables DOS and Windows application programs to use the real-time protected mode features of the iRMX Nucleus. Not all iRMX Nucleus system calls are supported by the RTE.

## RTE System Calls

A complete list of calls that are supported by both DOS and the iRMX OS is given in Table 2-1, which also lists the function code value for each RTE system call.

There are two additional calls that are only available from DOS. These calls, **rqe_read_segment** and **rqe_write_segment**, allow the application program to transfer data between DOS in VM86 memory and the iRMX OS in Protected Memory.

See also: **rqe_read_segment** and **rqe_write_segment** calls, *System Call Reference*

**CAUTION**

Some RTE calls are available from DOS, but not from Windows (and not from a DOS window in Windows). These include **rq_sleep** and all calls that support regions. Also, calls that block can only set a `time_limit` value of 0 or 1 when Windows is present. See the footnotes to Table 2-1 for details on calls that block.

**Table 2-1. RTE System Calls**

| Code | Call Name | Windows Limitation | Description |
|---|---|---|---|
| 0 | **create_mailbox** | | Create an object or data mailbox |
| 1 | **delete_mailbox** | | Delete an object or data mailbox |
| 2 | **send_message** | | Send an object to a mailbox |
| 3 | **send_data** | | Send data to a mailbox |
| 4 | **receive_message** | blocking | Receive an object |
| 5 | **receive_data** | blocking | Receive data |
| 6 | **create_semaphore** | | Create a semaphore |
| 7 | **delete_semaphore** | | Delete a semaphore |
| 8 | **send_units** | | Send a unit to a semaphore |
| 9 | **receive units** | blocking | Receive a unit from a semaphore |
| 10 | **create_region** | N/A | Create a region |
| 11 | **delete_region** | N/A | Delete a region |
| 12 | **send_control** | N/A | Release control of a region |
| 13 | **receive_control** | N/A | Receive control of a region |
| 14 | **accept_control** | N/A | Accept control of a region |
| 15 | **create_segment** | | Create a segment |
| 16 | **delete_segment** | | Delete a segment |
| 17 | **get_size** | | Get the size of a segment |
| 18 | **rqe_get_address** | | Get the physical address of a segment |
| 19 | **rqe_create_descriptor** | | Create a PVAM descriptor |
| 20 | **rqe_delete_descriptor** | | Delete a PVAM descriptor |
| 21 | **rqe_change_descriptor** | | Change a PVAM descriptor |
| 22 | **catalog_object** | | Catalog an object |
| 23 | **uncatalog_object** | | Uncatalog an object |
| 24 | **lookup_object** | blocking | Lookup an object |
| 26 | **get_task_tokens** | | Get task or job token |
| 27 | **get_type** | | Get the type of an object |
| 28 | **sleep** | N/A | Sleep for a specified time |
| 30 | **rqe_read_segment** | | Read from PVAM to a Real Mode segment |
| 31 | **rqe_write_segment** | | Write to a PVAM segment from a real mode segment |

blocking: These calls can only have a time_limit value of 1 or 0 when Windows is present. Otherwise they return an E_PARAM condition code.

N/A: These calls are only available from DOS, not when Windows is present. If invoked from Windows, they return an E_NOT_CONFIGURED condition code.

Generally, the RTE system calls allow the DOS application program to manipulate iRMX objects, semaphores, mailboxes, regions, segments and Protected Virtual Address Mode (PVAM) descriptors, and to communicate with iRMX tasks.

See also: VM86 Protected Mode Extensions, Chapter 3

If the DOS application program invokes an RTE system call that creates an iRMX object (such as a mailbox or PVAM descriptor), it must delete the iRMX object with the corresponding DOS RTE delete call from DOS. If the DOS application program does not explicitly delete the object, the object will be deleted upon termination of the DOS application program, or upon DOS being restarted.

The syntax and the semantics of the parameters for RTE system calls 0 to 28 are the same as the iRMX Nucleus system calls of the same name except for pointer parameters. The RTE system calls can return condition codes not returned by their Nucleus counterparts.

See also: System call descriptions, *System Call Reference*

All pointers parameters in these calls must be Real Mode pointers. Real Mode pointers consist of two 16-bit WORDs where the high WORD contains the base address of a 64 Kbyte segment and the low WORD contains the offset that points into the segment.

RTE is implemented by the VM86 Dispatcher in the DOS RTE system job. The DOS RTE job installs itself as a VM86 Protected Mode Extension at interrupt vector B8H. DOS RTE converts Real Mode pointers to PVAM pointers.

## RQEGetRmxStatus Call

Use the **RQEGetRmxStatus** RTE call to check if the iRMX OS is loaded. Use this call before any other RTE calls to insure RTE services are available. Unpredictable results occur if RTE calls are called when iRMX is not present.

See also: **RQEGetRmxStatus**, *System Call Reference*

The call returns `E_OK` if iRMX is loaded and running, or `E_EXIST` if iRMX is not present or unavailable. The call is provided in binary form as a linkable module in the file *\rmx386\demo\rte\lib\rmxfuncs.obj*.

## DLL for RTE Interfaces

iRMX for Windows includes a Dynamic Link Library (DLL), *rmx4win.dll*, which provides streamlined RTE interfaces to Windows programs. You must use the DLL with compilers such as Visual Basic in order to access the RTE calls. The RTE source and object code are installed with the iRMX for Windows software. You can modify this code and compile it.

## RTE Files

The C header file *\rmx386\demo\rte\lib\rmxintfc.c* contains all the declarations for the RTE functions.

When developing model DOS/Windows applications which make RTE calls, include the file *rmxintfc.h* and compile with the /AL switch (for Microsoft C). Use this file *rmxintfc.h* with all models of compilation. The file *rmxc.h* is specific to the RTE demo; do not use it in other applications.

There are three versions of the DOS RTE libraries: *dosrtec.lib* for compact model compilations, *dosrtes.lib* for small model compilations, and *dosrtel.lib* for large model compilations. The *\rmx386\demo\rte\lib* directory contains source for the libraries.

See also: *\rmx386\demo\rte\lib\readme.txt* file, for more information about the source code, header files, and libraries

# RTE Objects Limitation

The RTE job in iRMX for Windows maintains a table of all objects created by it on behalf of DOS/Windows applications. It handles up to 512 RTE-created iRMX objects, which is sufficient for most applications. However, this limit may be reached accidentally. When an RTE-created object is deleted in an iRMX task, the entry from the RTE table still remains. This causes the table to fill up with deleted objects.

The solution is to create a mailbox and send to it the tokens of objects to delete. Then have code in your DOS/Windows application which, when waiting for an iRMX event, queries the iRMX mailbox for objects to be deleted. If you delete all RTE objects that occur (which means iRMX applications receiving RTE-created objects need to send them to this mailbox for deletion), the RTE object table does not fill up and should then be able to handle all the active RTE objects needed by an application.

# Making an RTE System Call

All RTE system calls are accessed using a single software interrupt. The microprocessor registers and the Real Mode stack are used for passing the parameters of the RTE system call. One of the parameters passed is the RTE function code.

To invoke an RTE system call, the DOS application program must perform these actions:

1. Push all the parameters required by the RTE system call onto the Real Mode stack using the PL/M-286 convention. That is, the first parameter is pushed onto the stack, followed by the second and subsequent parameters.
2. Load the SI register to point to the last parameter pushed onto the Real Mode stack.
3. Load the AX register with the desired RTE function code.
4. Generate the RTE software interrupt request number, B8H.

This causes the RTE system call to execute and return control to DOS. When the DOS application program resumes, it must clear the parameters used by the RTE system call from the Real Mode stack.

If the DOS RTE system call returns a WORD (16-bit), it will be placed in the AX register. If it returns a DWORD, the high WORD will be placed in the DX register and the low WORD will be placed in the AX register.

DOS and its application programs run as an iRMX task under iRMX for Windows. If the application programs invoke RTE functions, they must obey the normal iRMX rules of not invoking the RTE from a hardware interrupt handler. In particular, the DOS TSR programs that typically hook themselves onto the hardware clock or keyboard interrupts must not issue RTE calls.

## Using RTE Functions

This example illustrates how a DOS application program can invoke one of the RTE functions. This example uses **rq_create_mailbox**. The example was compiled using Microsoft's assembler, MASM.

```
mov    ax,fifombx
push   ax                      ; PUSH 1st parameter - flags
mov    ax, SEG    exception    ; Get 2nd parameter into ES:AX
mov    es, ax                  ;
mov    ax, OFFSET exception    ;
push   es                      ; PUSH 2nd parameter -
push   ax                      ;    pointer to exception
mov    si,sp                   ; point si to last parameter
mov    ax,rqcreatembx          ; setup AX with FUNCTION CODE
int    0B8H                    ; CALL DOSRTE
add    sp,6                    ; remove parameters from stack
mov    mbx_tk,ax               ; save token
mov    ax,exception            ;
cmp    ax,eok                  ; check for validity
jne    error_p                 ;
```

# DOS RTE Demonstration

The DOS RTE demonstration program is menu-driven and enables you to exercise the RTE system calls at the DOS console. Two executable versions of the program are supplied: one runs as an iRMX application program, and the other runs as a DOS application program.

The source code and the executable for the demonstration programs are in the *\rmx386\demo\rte\obj* directory. The executable has two parts:

- *demo* is the iRMX part
- *demo.exe* is the DOS part

➡ **Note**

The DOS RTE demonstration program was compiled using Microsoft C, Version 7.0, compact model. If you are using the same compiler and model you can use the source as it is. Otherwise, compile the source using your compiler, make any necessary changes, and then recompile.

Both programs create, send, and delete iRMX objects, data, etc. The iRMX program makes iRMX system calls; the DOS program makes calls to the RTE system calls. Examine the demonstration program source code to see how the RTE system calls are invoked.

With one exception, the iRMX and DOS programs share the same source code, which has been compiled conditionally. This demonstrates how you can create your own application programs to run under either the DOS or the iRMX OS, and subsequently port them between the OSs.

The RTE system calls **rqe_read_segment** and **rqe_write_segment** are demonstrated by the Data Transfer (Real Mode/PVAM) functions of two different demonstration programs. The DOS version performs these functions by making RTE system calls; the iRMX program has code written specifically for this operation. This is necessary since the **rqe_read_segment** and **rqe_write_segment** system calls provided by the RTE are not required for the iRMX OS.

See also: **rqe_read_segment** and **rqe_write_segment**, *System Call Reference*

## Example: Running the Demonstration Program

To start the demonstration, change to the *\rmx386\demo\rte\obj* directory. If you are at an iRMX prompt, run the iRMX *demo* program. If you are at a DOS prompt, run the DOS *demo.exe* program. Both programs display this menu:

```
DOS/iRMX Real Time Extensions Demo Program
========================================

1. Mailboxes (Objects) Functions
2. Mailboxes (data) Functions
3. Semaphore Functions
4. Segment Functions
5. Descriptor Functions
6. Data Transfer Functions
7. Display Help on above functions
8. Exit (terminate program)

Enter option (1 to 8) :-
```

Press the <Alt +> and <Alt -> keys (using the plus and minus keys on the numeric keypad) to change the background and foreground colors for the iRMX version of the demonstration. Since the appearance of the menus is identical, you can use color to tell you whether you are in the DOS or the iRMX version.

See also: Changing iRMX Console Color, *Installation and Startup*

You may invoke six different types of functions: mailboxes for data and objects, semaphores, PVAM segments, descriptors, and data display. The functions are described in the following sections.

## Mailboxes (Objects) Functions

To invoke any of the Object Mailbox functions, enter:

**1<CR>**

in response to the Main Menu prompt. A menu similar to the Object Mailbox menu appears.

```
Object Mailbox Functions
========================

1. Send object to mailbox
2. Receive object from mailbox
3. RETURN to previous menu

Enter option (1 to 3) :-
```

These functions allow you to send and receive objects (segments or descriptors) to or from a named mailbox. The mailbox may be created by either the DOS or iRMX version of the demonstration program.

### Send Object to Mailbox

If the mailbox does not exist, it is created; if the named object does not exist, a segment is created for the object.

### Receive Object from Mailbox

If the received object is a segment, its name is displayed. Otherwise, the iRMX token for the object is displayed, or if there are no objects, an E_TIME condition code is displayed.

If the mailbox does not exist, an error message appears.

## Mailboxes (Data) Functions

To invoke any of the Data Mailbox functions, enter:

```
2<CR>
```

in response to the Main Menu prompt. A menu similar to the Object Mailbox appears.

Data mailbox functions send and receive a string of text (up to a maximum of 127 characters) to and from a data type mailbox. The mailbox may be created by the DOS or iRMX version of this program.

### Send Data to Mailbox

Enter the string at the prompt. The text entry must be terminated by a <CR>. If the requested data mailbox does not exist, one will be created.

### Receive Data from Mailbox

This option receives a string of text from the specified data mailbox and displays the text and the size of the text string on the screen. If the specified mailbox does not exist, an error message appears.

## Semaphore Functions

To invoke any of the Semaphore functions, enter:

**3<CR>**

in response to the Main Menu prompt. A menu similar to the Object Mailbox appears.

Semaphore functions send and receive units to and from a semaphore. The semaphore may be created by either the DOS or the iRMX version of this program.

### Send Units to Semaphore

The semaphore will accept a maximum of 10 units. When prompted, enter the number of units to send.

### Receive Units from a Semaphore

This option receives a requested number of units from a named semaphore and displays the remaining number of units at the semaphore. If the semaphore does not exist, an error message appears.

## PVAM Segment Functions

To invoke any of the PVAM Segment functions, enter:

**4<CR>**

in response to the Main Menu prompt. This menu appears:

```
PVAM Segment Functions
=====================

1. Create PVAM Segment
2. Delete PVAM Segment
3. Display PVAM Segment
4. RETURN to previous menu

Enter option (1 to 4) :-
```

If you are not creating a segment, you can delete or display a segment created previously by either the DOS or the iRMX version of this program.

### Create PVAM Segment

This option creates a named PVAM segment of any size. You can use this segment as either the source or destination of a copy operation to or from Real Mode memory. You can also pass the PVAM Segment to object mailboxes as well as display them by the Display PVAM Segment function.

### Delete PVAM Segment

This option deletes a named PVAM segment or named descriptor. If you created the segment from DOS, delete it from DOS.

### Display PVAM Segment

This option displays a PVAM segment or descriptor in blocks of 160 bytes maximum. The PVAM segment or descriptor displays in lines of 16 bytes, followed by the printable ASCII characters for each byte. If a byte is not a printable ASCII character, a . (period) is displayed instead. You are prompted for input to continue (any key) or quit (Q or q).

## Descriptor Functions

To invoke any of the Descriptor functions, enter:

```
5<CR>
```

in response to the Main Menu prompt. A menu similar to the PVAM Segment menu appears.

If you are not creating a descriptor, you can delete or display a descriptor created previously by either the DOS or iRMX version of this program.

### Create Descriptor

This option creates a named descriptor of any size and absolute address.

### Delete Descriptor

If you created the descriptor from DOS, delete it from DOS.

### Display Descriptor

This option displays a PVAM segment or descriptor in blocks of 160 bytes maximum. The PVAM segment or descriptor is displayed in lines of 16 bytes, followed by the printable ASCII characters for each byte. If a byte is not a printable ASCII character, a . (period) is displayed instead. You are prompted for input to continue (any key) or quit (Q or q).

The segment or descriptor is looked up under its user name.

### Data Transfer Functions

To invoke any of the Data Transfer (Real Mode/PVAM) functions, enter:

```
6<CR>
```

in response to the Main Menu prompt. This menu appears:

```
REAL MODE/PVAM Copy Functions
=============================

1. Copy PVAM segment to real mode address
2. Copy Real mode address to PVAM segment
3. RETURN to previous menu

Enter option (1 to 3) :-
```

#### Copy PVAM Segment to Real Mode Address

You are prompted for the Real Mode segment and offset.

**CAUTION**

Do not copy data over vital DOS system or application memory, or to memory mapped out to I/O devices. Otherwise, your system could develop problems.

#### Copy Real Mode Address to PVAM Segment

This option copies a specified Real Mode address to a specified PVAM Segment. You are prompted for the Real Mode Segment and Offset and also the PVAM Segment and Offset.

## Example: WTERM Demonstration Program

WTERM is a demonstration terminal emulator Windows program. This program uses DOS RTE to display the iRMX command line within a window. You can invoke any iRMX commands while this demonstration program is running.

The *\rmx386\demo\wterm* directory contains source code for the program and the *\dosrmx* directory contains the executable code.

See also: Running WTERM, *Installation and Startup*

# VM86 Protected Mode Extensions 3

The VM86 Dispatcher enables you to write Protected Virtual Address Mode (PVAM) extensions for DOS. These extensions are also known as VM86 Protected Mode Extensions. These extensions allow DOS application programs running in VM86 Mode to change to Protected Mode, obtain Protected Mode services, and then return to VM86 Mode.

All VM86 Protected Mode Extensions are implemented as software interrupt handlers using the software interrupt instruction. The VM86 Dispatcher in Protected Mode intercepts all software interrupt requests. To run a VM86 Protected Mode Extension, the VM86 Dispatcher calls the required interrupt handler to service the particular request, and then returns to DOS in VM86 Mode. If the VM86 Dispatcher intercepts an interrupt request which is not a VM86 Protected Mode Extension request, that interrupt request is reflected back to DOS.

The RTE described in the previous chapter is an example of a VM86 Protected Mode Extension.

## Installing VM86 Protected Mode Extensions

Each VM86 Protected Mode Extension you write, though implemented as an iRMX program, is invoked when a DOS application program issues an appropriate software interrupt. Each extension must be installed at a unique interrupt level and an extension may contain a number of subfunctions, as does the RTE. You can choose the method of passing the extension's subfunction. The RTE uses the AX register to hold the function's code.

Install the extension at its desired interrupt level using the **rqe_set_vm86_extension** system call.

See also: **rqe_set_vm86_extension**, *System Call Reference*

**Note**

VM86 Protected Mode Extension cannot be used from within Windows running in Standard Mode.

## iRMX Interrupt Levels

Table 3-1 lists the interrupt levels in the Interrupt Descriptor Table (IDT) used by the iRMX for Windows OS.

**Table 3-1. iRMX Interrupt Levels**

| Interrupt | | Function |
|---|---|---|
| **Hex** | **Decimal** | |
| 00H-10H | 0-16 | Microprocessor traps and DOS hardware vectors |
| 11H-20H | 17-32 | *ROM BIOS services |
| 21H-2FH | 33-47 | DOS services |
| 38H-3FH | 56-63 | iRMX hardware vectors for Master PIC |
| 50H-57H | 80-87 | iRMX hardware vectors for Slave PIC |
| 5BH | 91 | Network Redirector |
| 80H | 128 | Used by the VM86 Dispatcher |
| 85H | 133 | iRMX Interface TSR, supports chaining however |
| B8H | 184 | DOS RTE |
| C3H | 195 | UDI |

* You may install extensions to monitor or evaluate these calls.

Interrupts and ranges not listed in Table 3-1 are available for user-written extensions.

To install an extension, call **rqe_set_vm86_extension** and pass it these parameters:

1. The desired interrupt level for the extension.
2. The entry point for the extension itself. This entry point defines where the extension is located in system memory so that it may be invoked when DOS makes the appropriate interrupt request.

3. The entry point for the extension's deletion handler. The deletion handler is not mandatory, but each extension can have one. Any extension which is used by DOS to create iRMX objects should have a deletion handler to delete those objects when the DOS program terminates.
4. A pointer to a WORD (16-bit) in system memory which the VM86 Dispatcher uses to return a status code for this call.

Once an extension has been installed, it remains active until it is deactivated with the **rqe_set_vm86_extension** system call. Call **rqe_set_vm86_extension** again and pass it the same parameters, but with the VM86 Extension Entry pointer set to null.

## Extension Procedure Operation: DOS Interrupt Handling

Interrupts generated by DOS in VM86 mode are vectored to the PVAM handler referenced in the processor's IDT. The VM86 Dispatcher invokes a particular extension in response to an interrupt received at the `int_level` specified in the **rqe_set_vm86_extension** system call.

All DOS interrupts are intercepted by the VM86 Dispatcher and are processed as follows:

1. If an interrupt requires a Real Mode handler installed by DOS, the VM86 Dispatcher deflects the interrupt back to that Real Mode interrupt handler.
2. If the interrupt requires a PVAM interrupt handler, the Dispatcher enables the interrupt handler to run; the DOS application program is running in VM86 mode and all VM86 Mode-generated interrupts naturally vector to the PVAM interrupt handler in the IDT. The interrupt handler returns control back to the DOS application program upon termination.
3. If the interrupt requires a VM86 Extension, the VM86 Dispatcher calls the entry point of the extension. The extension then executes and returns to the VM86 Dispatcher, which then returns control back to the DOS application program that made the interrupt. The VM86 Dispatcher calls the VM86 Extension, and passes to it a pointer to a structure defining the DOS machine state and a value defining the context of the DOS interrupt handler. The VM86 Dispatcher expects the extension to return a byte indicating that the request has been processed completely.

   See also: **rqe_set_vm86_extension**, *System Call Reference*

## Deletion Handler Operation

The VM86 Dispatcher calls all extension deletion handlers when any DOS program is deleted. Any of these conditions can delete a DOS program:

- When a DOS application program terminates using DOS system calls INT 20H or INT 21H
- When a <Ctrl-C> is typed in the middle of a DOS program, and the program has not changed the default <Ctrl-C> handler in DOS

All installed deletion handlers are called sequentially by the VM86 Dispatcher. The VM86 Dispatcher calls the deletion handler with a flag that indicates:

- If the current DOS program is being deleted
- If all DOS programs are being deleted

This helps the VM86 Dispatcher perform the appropriate cleanup. For example, if the VM86 extension has created iRMX objects, the deletion handler will know which objects to delete.

**Note**

Every DOS program has a unique identifier: the address of its Program Segment Prefix (PSP). The VM86 extension can use **rqe_dos_request** to obtain the current PSP. This enables the VM86 extension to track which resources are allocated to which DOS program.

To ensure that the PSP address obtained is the PSP of the current DOS program, rather than that of the RMX TSR program, set the `tsr_flag` parameter to 1 in the **rqe_dos_request** call.

## Extension System Call Restrictions

The extensions called by the VM86 Dispatcher can use only system calls in the BIOS and Nucleus subsystems of the iRMX OS. Extensions run in the context of the VM86 Dispatcher Job, and can only make the same system calls as the Dispatcher Job.

## Extension Installation Examples

This section discusses three code segments which illustrate:

- Installing an extension from the iRMX OS
- Two ways of initiating an extension from DOS

### Installing an Extension from the iRMX Operating System

An iRMX program *\rmx386\demo\c\vm86ext\rmxext.c* illustrates how to install an extension. It also gives example code for both the VM86 Extension entry procedure and the deletion handler.

In the example, `main()` creates a mailbox, prints an installation message, and installs the extension for interrupt level 0B9H (185 decimal). It then waits to receive a message at the mailbox. At this point, `main()`waits until the DOS part of the example issues INT 0B9H.

When the DOS part issues INT 0B9H, `main()` calls the extension procedure. The `entry_procedure` extension procedure sends a message to the mailbox created in `main()`, where `main()` is waiting. `Main()` receives the message, prints it out, and deletes in order, the segment described by `segtoken`, the mailbox, and the extension.

The example was compiled and bound using Intel's iC-386 compiler and BND386 binder. The mailbox token, `mbxtoken`, is an iRMX object.

### Initiating an Extension from DOS

The DOS part of the example uses two programs. The C program is in *\rmx386\demo\c\vm86ext\dosext.c*, and the assembly language program is in *\rmx386\demo\c\vm86ext\dosext.asm*. These two programs issue INT 0B9H (185 decimal). Both examples have the same functionality.

The DOS application C program illustrates one way the previously installed extension can be initiated. The example was compiled and linked using the Microsoft Version 7.0 C compiler.

The DOS application assembly program illustrates one way the previously installed extension can be initiated. The example was created using the Microsoft Version 5.1 assembler.

# Making DOS and ROM BIOS System Calls 4

This chapter describes how to make DOS and ROM BIOS system calls from iRMX application programs.

## Making DOS and ROM BIOS Calls from an iRMX Application

The **rqe_dos_request** system call enables the iRMX OS to make ROM BIOS and DOS requests in much the same way as the RTE system calls allow a DOS application program to make iRMX system calls.

Using **rqe_dos_request**, a DOS application program can be ported to an iRMX environment to take advantage of the Protected Mode features without changing all DOS and ROM BIOS calls to iRMX system calls.

The application program can also use the **rqe_dos_request** system call to access a DOS device driver which may be running in VM86 Mode. However, the application program must not use a DOS system call that conflicts with the file server.

The DOS data structure represents the microprocessor registers, and the **rqe_dos_request** system call passes a pointer to this structure.

See also: **rqe_dos_request** and DOS data structure, *System Call Reference*

To make DOS/ROM BIOS calls, the application program must set the appropriate register values in the structure pointed to by `register_ptr`, set the `int_num` parameter with the required DOS interrupt level, and set the `xfer_data` byte, the source and destination transfer pairs, pointers, and counts based on the data being transferred. The **rqe_dos_request** system call can then be invoked and the required DOS system call will be made. The WORD pointed to by the `status_ptr` parameter contains the condition code generated by the **rqe_dos_request**. If the call was successful, the structure pointed to by the `register_ptr` parameter reflects the register values returned by the DOS system call.

Many DOS system calls pass data. This can be done by passing the segment base and offset of the source or destination address, where data is located in one or more register pairs. Other registers sometimes specify the length of data located at the address specified by the register pair.

The DOS system call **Get Redirection List Entry**, Interrupt 21H, Function 5FH, Subfunction 02H, uses the DS:SI register pair to point to a 16-byte (maximum) character string containing a device name in the redirection list. The ES:DI register pair points to a 128-byte (maximum) character string containing the network name of the device. The data transfers from the system call to the calling application program.

To make a similar call using the **rqe_dos_request** system call, you use four separate sets of structure elements to control the data transfer. These structure elements indicate the appropriate registers, but are ignored if the `xfer_data` structure element is set to 0.

See also: **rqe_dos_request**, *System Call Reference*

## Example: Get Free Disk Space

The **Get Free Disk Space** DOS system call transfers data only in microprocessor registers. To make the DOS system call **Get Free Disk Space** (of a DOS drive) Interrupt 21H, Function 36H, the iRMX application would:

1. Set `int_num` (DOS system call interrupt number) to 21H.
2. Set `reg_ah` (Interrupt 21H subfunction code) to 36H.
3. Set `reg_dl` (Drive code) to the required drive code level where 1 = drive A, 2 = Drive B, etc.
4. Set `xfer_data` to 0, as no data is transferred with this system call except directly using the microprocessor registers.
5. Set `status_ptr` to point to a WORD variable, which the **rqe_dos_request** system call sets with a condition code before returning to the iRMX application program.
6. Make the **rqe_dos_request** system call.

See also: **rqe_dos_request**, *System Call Reference*

To determine the result of the requested DOS system call, the application program would then:

7. Determine the value of the WORD variable pointed to by `status_ptr`. If the **rqe_dos_request** call did not succeed (condition code not equal to E_OK), the application program terminates with an appropriate error message.
8. If the condition code returned was 0, the application program could proceed.
9. The values stored in `reg_al` and `reg_ah`, on return from the call, hold these values:

| | |
|---|---|
| `reg_ah` = FFH<br>`reg_al` = FFH | DOS determined that the drive code specified by `reg_dl` was not valid. |
| or | |
| `reg_ah`<>FFH<br>`reg_al`<>FFH | `reg_ah`, `reg_al` specifies the sectors per cluster of the specified drive. |

10. If the drive code was valid, the other register values are set as:

| | |
|---|---|
| `reg_bh`,<br>`reg_bl` | Specifies the number of available clusters on the specified drive. |
| `reg_ch`,<br>`reg_cl` | Specifies the number of bytes per sector on the specified drive. |
| `reg_dh`,<br>`reg_dl` | Specifies the number of clusters (used or available) on the specified drive. |

## Get Redirection List Entry Example

The example *\demo\c\vm86ext\dosdevs.c* uses the DOS system call **Get Redirection List Entry**. The first part shows to set registers in the DOS data structure prior to making the **rqe_dos_request** system call. The second part shows how to make the **rqe_dos_request** system call. The example was compiled and bound using Intel's iC-386 compiler and BND386 Binder.

### Setting the DOS Data Structure

The DOS system call **Get Redirection List Entry**, Interrupt 21H, Function 5FH, Subfunction 02H, uses the DS:SI register pair to point to an ASCIIZ (null-terminated) character string defining the local device name found in the list, and uses the ES:DI register pair to point to the ASCIIZ character string defining the network name for that local device.

For this example, though no source data is transferred, two character strings are created by the DOS system call destination data parameter, which are pointed to by the two register pairs.

Set the DOS data structure as follows:

1. Set `int_num` (DOS system call interrupt number) to 21H.
2. Set `tsr_flag` to 0.
3. Set `reg_ah` (Interrupt 21H function code) to 5FH.
4. Set `reg_al` (Function 5FH subfunction code) to 02H.
5. Set `reg_bx` to the required redirection list index.
6. Set `xfer_data` to 0FFH since data is transferred with this system call.
7. Set `src1_xfer_pair` and `src2_xfer_pair` to 0 since no source data transfer is required.
8. Set up the destination data control parameters for the local device name as follows:

| | |
|---|---|
| `dest_p_1` | &local_device_name (local_device_name is a character array) |
| `dest1_xfer_pair` | 4 (to specify the DS:SI register pair) |
| `dest_count_1` | 16 (to specify a maximum size of 16 characters) |

9. Set up the destination data control parameters for the network name as follows:

| | |
|---|---|
| `dest_ptr_2` | &network_name (network_name is a character array) |
| `dest2_xfer_pair` | 8 (to specify the ES:DI register pair) |
| `dest_count_2` | 128 (to specify a maximum size of 128 characters) |

10. Set `status_ptr` to point to a WORD variable which the **rqe_dos_request** system call will set with a condition code before returning to the iRMX application.

11. Make the **rqe_dos_request** system call.

    The **rqe_dos_request** system call returns a value in the WORD variable pointed to by `status_ptr`. The meanings of the values are:

| Value | Meaning |
|---|---|
| Not 0 | The call encountered an error, as specified by the error code. The iRMX application needs to evaluate the error to see if a retry is possible. |
| 0 | Since no errors were encountered by the iRMX OS, the application can proceed. |

    For certain DOS system calls, including this example, the carry flag is set to one of two values.

| Value | Meaning |
|---|---|
| 1 | The DOS system call failed. If the call failed, `reg_al` will contain the DOS error code. |
| 0 | The DOS system call was successful. |

    In this example, if the DOS system call is successful, these parameters return to the caller from the DOS system call:

| | |
|---|---|
| `reg_bh` | Device status flag |
| `reg_bl` | Device type |
| `reg_cx` | Stored parameter value |

# General Information 5

This chapter describes various information related to iRMX for Windows. The information includes such areas as programming techniques, use of iRMX objects, and network concepts. This information does not apply to the iRMX III or iRMX for PCs OSs.

## Interrupt Virtualization and Determinism

iRMX for Windows has two interrupts modes in which DOS and Windows can operate. You can configure the mode by changing the appropriate setting in the *\rmx386\config\rmx.ini* file. These modes are Interrupt Virtualization Enabled (VIE=0FFH) and Interrupt Virtualization Disabled (VIE=00H). The default is Interrupt Virtualization Disabled.

With Interrupt Virtualization disabled, DOS executes all real mode instructions supported by the microprocessor, including ENABLE and DISABLE INTERRUPTs. This affects system performance in interrupt latency and determinism by enabling DOS and ROM BIOS-based disk I/O to operate at near optimum DOS performance levels. In this mode, the iRMX OS has little interaction with DOS and Windows.

With Interrupt Virtualization enabled, the iRMX OS traps all DOS and Windows access of privileged instructions (CLI, STI, INT, POPF, etc) as well as attempts to access the Programmable Interrupt Clock and Programmable Interrupt Timer. The iRMX OS can virtualize interrupts with respect to DOS/Windows, while keeping interrupts disabled for the iRMX OS for as short a time as possible. This increases interrupt response time and decreases interrupt latency for iRMX-owned interrupt levels, such as non-DOS levels. However, this frequent intervention by the iRMX OS into DOS/Windows operations also affects DOS and ROM BIOS-based I/O performance. Small transfers slow down dramatically while larger transfers (4 Kbytes or larger) experience relatively little degradation.

Based on the needs of the application, iRMX for Windows can be optimized for either higher performance DOS/ROM BIOS-based I/O with less than ideal determinism or for solid determinism with less than ideal DOS/ROM BIOS-based I/O performance. To get both solid determinism and high performance I/O, first add an iRMX-owned disk controller such as the Adaptec 1542, then use the PCI loadable driver provided in the product, and finally set VIE=0FFH in the *rmx.ini* file.

See also: *rmx.ini* file, *System Configuration and Administration*

## Real-time Fence

The real time fence is set at priority level 127. All active DOS-owned interrupts are temporarily masked when tasks are running at or above (numerically lower than) priority level 127. The real-time fence here is different from the real-time fence used with round robin scheduling. This preserves the real time aspect on the iRMX OS side of an iRMX for Windows application system. If you make an **rqe_dos_request** call from an iRMX task running at a priority above or equal to this real-time fence, you will receive an E_TIME condition.

See also: Real-time fence, *System Configuration and Administration*

## iRMX-NET Access From a DOS Server

You can access a DOS server indirectly from the iRMX for Windows side using the EDOS File Driver. For example:

From the DOS side of iRMX for Windows, do this:

```
net use r: \\<dos-server>
dir r:
```

Then, from the iRMX side of iRMX for Windows, do this:

```
ad r_dos as :r: e
dir :r:
```

See also: Using iRMX-NET in a DOS Environment, *System Configuration and Administration*

If you try to directly access a DOS server from the iRMX OS, such as with the **attachdevice** command, you may encounter a General Protection fault or similar failure.

# Remote Launch 6

In addition to discussing the Remote Launch facility, this chapter also describes network information related to iRMX for Windows. This information covers such areas as file exchange, loading sequences, and network support.

## Invoking iRMX Programs from DOS or Windows

An iRMX OS can launch (invoke) iRMX applications on a remote system from either a local DOS-running or Windows-running system. The files to do this are:

*rls.job* - iRMX Remote Load Server (RLS) job (in the *\rmx386\jobs* directory)

*dlc.exe* - DOS Load Client (in the *\dosrmx* directory)

*wlc.exe* - Windows 3.1 Load Client (in the *\dosrmx* directory)

Load *rls.job* from the iRMX OS. This system can be physically remote, such as a networked Multibus II system, or can be the same system that runs DOS/Windows.

The files *dlc.exe* or *wlc.exe* invoke iRMX applications on an iRMX partition or system from a DOS or Windows system, respectively.

Figure 6-1 shows a network configuration where PC clients can launch an iRMX application on a remote Multibus system. Figure 6-2 shows how a DOS application can launch an iRMX application in the same system.

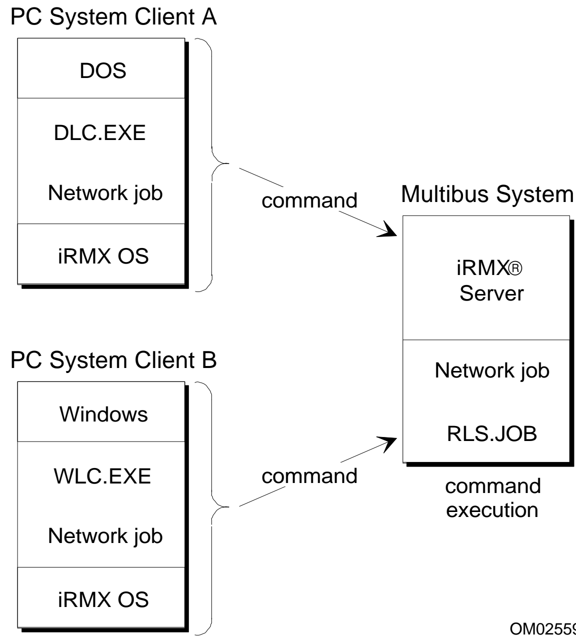



**Figure 6-1.  Launching Commands from Multiple Clients to a Single Server**

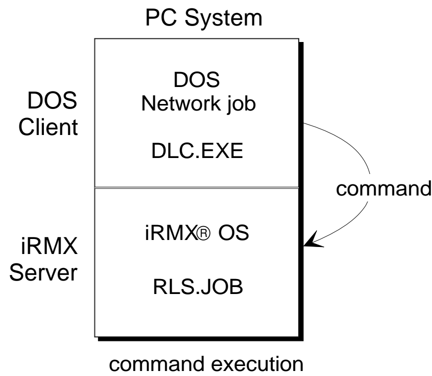



**Figure 6-2.  Launching Commands within a Single System Client and Server**

# Using the iRMX Load Server with iRMX for Windows

For DOS Load Clients (DLC) and Windows Load Clients (WLC) to communicate with the iNA 960 networking, the iRMX Load Server's system must be have an RLS network name established. In iRMX for Windows systems (Figure 6-2), the iRMX network redirector job is required as well.

Use the **setname** command to establish the RLS property type for the server if you do not have a network card. Use the **loadname** command and the **setname** command to establish the RLS property type for the server if you have a network card. The **loadname** command does not establish the RLS property type if a MIP job is loaded.

See also: **loadname, setname** commands, *Command Reference*,
Loadable Jobs and Drivers, *System Configuration and Administration*

Use this procedure to load the iNA 960 files (in addition to these, you can also load iRMX-NET jobs):

1. Load and log on to the iRMX OS.
2. Load the appropriate network job using the **sysload** command. This example uses the network job for a PC system without a network card.

```
- sysload /rmx386/jobs/inl*n.job <CR>
```

See also: *i*.job*, *System Configuration and Administration*,
**sysload** command, *Command Reference*

3. Load the *netrdr.job* as follows:

```
- sysload /rmx386/jobs/netrdr.job <CR>
```

4. Load the *rls.job* file as follows:

```
- sysload /rmx386/jobs/rls.job <CR>
```

5. Using the **setname** command, assign a remote server name. This example assigns a name of “remote”.

```
- setname remote <CR>
```

6. Toggle to the DOS prompt using <Alt>-<SysReq>.
7. Run *pcnet.exe* at the DOS prompt.

The DOS and iRMX systems are now initialized for remote launching. Refer to the section on the *dlc.exe* file to launch applications from the DOS prompt or refer to the section on *wlc.exe* to launch applications from the Windows prompt.

# Using MS-NET with the PCLINK2

For DOS Load and Windows Load Clients to communicate with the iRMX Load Server, the iRMX Load Server's system must be running one of the iNA 960 networking jobs with an RLS network name established. The clients must be running MS-NET (*netbios.exe*). Figure 6-3 shows an example of these systems.

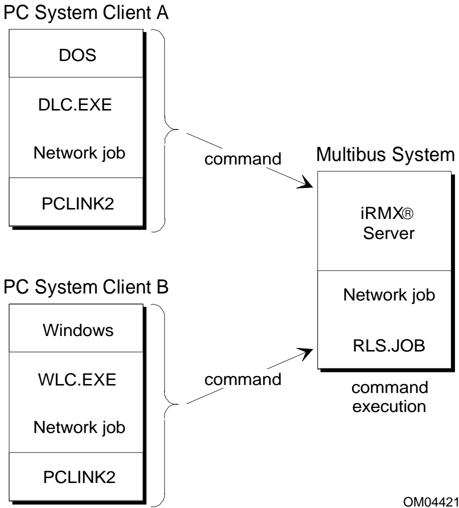



**Figure 6-3. DOS-only System Accessing an iRMX System**

To access an iRMX Load Server from a DOS client using *netbios.exe* with a PCLINK(2) communication board, you must modify the *config.nia* file. Use this procedure:

1. At the DOS prompt, change to the directory where the PCLINK software is loaded. This directory should contain the *config.nia* for MS-NET *netbios.exe*. If this directory does not exist, change to the *\pclr3* directory.
2. Invoke *admin.exe*.
3. Choose menu item 3, Modify Socket Type Table.
   A. Press the <Up arrow> twice to move the cursor to the "Type PgDn..." prompt.
   B. Press the <PgDn> key until socket 52H appears. Use the <Up arrow> key to move into the field to edit it.
   C. Change the type value for socket 52H to 0006.
   D. Use the <PgDn> key and change the type value for socket 72H to 0006.

4. Use the <PgUp> key to move to the "Update File" option. Update the *config.nia* file with a "Y".
5. Exit the *admin.exe* program using the <Esc> key.

The remote and local systems are now initialized for remote launching. Refer to the section on the *dlc.exe* file to launch applications from the DOS prompt or refer to the section on *wlc.exe* to launch applications from the Windows prompt.

See also: Loadable Jobs and Drivers, *System Configuration and Administration*

# Using RLS.JOB

The *rls.job* file, in the */rmx386/jobs* directory, acts as a Remote Load Server. It lets you launch iRMX applications remotely for execution on the Remote Load Server's system.

⟹ **Note**
The Network Redirector job will also need to be loaded if you wish to launch iRMX programs from DOS or Windows on the same system. It is not required if programs are only to be launched from remote clients.

See also: *i*.job*, Loadable Jobs and Drivers,
*System Configuration and Administration*

## Running RLS.JOB

By default, the **sysload** invocation of this job is commented-out in the file *:config:loadinfo*. To automatically load *rls.job*, uncomment (remove the semicolon) from the *rls.job* entry in the file *:config:loadinfo*. You can also manually load *rls.job* by using the **sysload** command from the iRMX prompt as follows:

```
- sysload /rmx386/jobs/rls.job [MAX_DIALOGS=nnn] [MAX_MESSAGES=mmm]
```

Where:

MAX_DIALOGS (MD) is an optional parameter that specifies the maximum number of simultaneous client dialogs supported by this server (1 to 255; default is 16)

MAX_MESSAGES (MM) is an optional parameter that specifies the maximum number of simultaneous outstanding terminal output messages supported by this server (default is 256).

# Using DLC.EXE

The file *dlc.exe* is a DOS executable installed in your *\dosrmx* directory. Use it to launch iRMX programs from the DOS command line. The *dlc.exe* file requires MS-NET or the iRMX-NET network redirector.

After starting MS-NET or the iRMX-NET network redirector, invoke *dlc.exe* with this syntax:

**`C:\> dlc [-B] [-F`*`filename`*`]`** ***`server`*** **`[`*`dirpath`*`]`** ***`command`***

`-B` Batch mode to support output redirection;
i.e., **`dlc -B srvr plm386 prog.plm > logfile`**

`-F`*`filename`*
Initial Esubmit file name (executed before command)

*`server`* iRMX Load Server network name (RLS property type)

*`dirpath`* iRMX directory path to the executable command (use forward slashes i.e., `/rmx386/demo/c/intro/demo`)

*`command`* iRMX command (and any parameters) to be executed

# Using WLC.EXE

The file *wlc.exe* is a Windows executable installed in your *\dosrmx* directory. Use *wlc.exe* to launch iRMX programs from Windows. The *dlc.exe* file requires MS-NET or the iRMX Network Redirector.

## Creating the WLC Icon

To add a Windows icon for *wlc.exe*, do this:

1. Run Windows.
2. Pull down the FILE menu and select the NEW option.
3. Select PROGRAM ITEM and click OK. A Program Item Properties box appears.
4. In the Description field, enter “WLC” or some other descriptor.

5. In the Command Line field, enter “\DOSRMX\WLC.EXE”.
6. Click OK. An icon should appear in your Windows Application Group window.

## Running WLC.EXE

To run *wlc.exe*, do this:

1. Start MS-NET or the iRMX-NET network redirector and then start Windows.
2. Start *wlc.exe* by double-clicking on the WLC icon. When the application starts to run, a window will appear. From this window, pull down the PROGRAM menu and select the LOAD option.
3. A dialog box will pop up asking for the server name, the initial Esubmit file name, and the command name.
    A. Enter the iRMX-NET network name (RLS property type) for the iRMX Load Server.
    B. Optionally, enter the name of an Esubmit file that is to be executed before the command is.
    C. Enter the directory path and the file name of the iRMX application to be executed.
4. Click the OK button.

To pre-configure any of these parameters in *wlc.exe*, do this:

1. Single click on the WLC icon.
2. Pull down the File menu and select the Properties option. Edit the Command Line as follows:

```
wlc [-A] [-Ffilename] [-Hnnn] [-Wnnn] [-Xnnn] [-Ynnn] [-E]
    [server [dirpath command]]
```

Where:

`-A` Aborts on exit instead of leaving a completed window

`-Ffilename` Initial Esubmit file name (executed before command)

`-Hnnn` the height of window

`-Wnnn` Width of window

`-Xnnn` X position of upper left-hand corner of window

| | |
|---|---|
| -Y*nnn* | Y position of upper left-hand corner of window |
| -E | Enables support for the IBM-extended ASCII character set. This support includes multiple-byte scan codes, such as <Alt>-<F1>, and the display of special graphics characters |
| *server* | iRMX Load Server network name (RLS property type) |
| *dirpath* | iRMX directory path to the executable command (use forward slashes) |
| *command* | iRMX command (and any parameters) to be executed |

If the server and the command are not both specified on the invocation line, specify them in the dialog box for the Program Load menu option.

**CAUTION**

The Esubmit file must not include an invocation of the iRMX CLI command. Including the CLI command in the Esubmit file causes unpredictable results. To invoke the iRMX CLI command from WLC, specify it as the command.

By loading the iRMX CLI command from DLC or WLC, an iRMX user session is created. You can create multiple iRMX user sessions by multiple invocations of WLC.

# Using DDE 7

This chapter describes how to use the iRMX Dynamic Data Exchange (DDE) facility to communicate between Windows-based graphical user interfaces (GUIs) and iRMX real-time applications. It is organized into conceptual information, reference information, and examples. This chapter is written for the programmer who already understands DDE concepts.

See also: Your Windows programming manual for more information on DDE messages

## Real-time DDE Capabilities

Dynamic Data Exchange (DDE) is a standard inter-process communications protocol for Windows. DDE enables programs to easily and freely exchange data on a one-time or continuous report-by-exception basis. Network DDE, or NetDDE as it is better known, is a class of facilities that extends the Windows DDE capabilities transparently across networks and other communications facilities. It is accomplished by adding one or more DDE Message Routers to the Windows product. The Message Routers function as proxy agents for remote clients and servers.

The iRMX NetDDE facilities are based upon a set of custom DDE Message Routers that route DDE messages between Windows and iRMX systems using various networking and communications protocols. On the Windows system, a Router is a stand-alone executable program that can be invoked either as part of the Windows automated start-up procedure or manually when its facilities are needed. On the iRMX system, the Router is a library of routines that, when bound with a real-time application, provides it with the necessary routing capabilities.

## DDE Terminology

These DDE terms are used extensively in the rest of this chapter, especially when discussing DDE interaction between applications.

| | |
|---|---|
| client | An entity that establishes communications with one or more servers and requests data from, or sends data to, a server. An iRMX application can be a server for one conversation and a client for another. |
| server | An entity that maintains a set of data and can make this data available to one or more clients. |
| conversation | A link between a client and a server that the client uses to obtain access to a specific subset of the data. The data is specified by a topic name at the time the conversation is established. |
| application name | The name of the server application. For example, if the server application is the Excel spreadsheet program, the application name is *Excel*. The application name is also known as the service name. |
| string types | Strings in DDE messages are null-terminated (ASCIIZ) strings. They are different from iRMX strings, where the first byte is the string length. |
| topic name | The general classification of data items that can be exchanged during a DDE conversation. This is usually either a filename if the conversing applications exchange files, or an application-specific name. |
| data item | The actual information related to the topic name. Values for data items are exchanged during conversations. |
| cold link | The client routinely polls the server to obtain the value of a data item. The value can change at any time but the server does not notify the client of the change. |
| warm link | The server notifies the client when the value of a data item changes. The server sends the new value only when the client requests it. |
| hot link | The server notifies the client when the value of a data item changes and immediately sends the new value to the client. |

## Communications Protocols

The iRMX NetDDE facility supports these communications protocols:

- TCP/IP (*routetcp.exe* and *tcpdde.lib*) - Use this networking protocol between iRMX systems and any version of Windows that supports the Winsock interface (using *winsock.dll*) and provides Ethernet or SLIP network access. These capabilities are built into Windows NT, Windows 95 and Windows for Workgroups (when the Microsoft TCP/IP for Windows for Workgroups add-on package is present). You can also use Windows 3.1 or 3.11 if you add a compatible TCP/IP add-on package.
- OpenNET (*routenb.exe* and *inadde.lib*) - You can use this networking protocol in any of the following environments:
  - Between iRMX systems and PCs running Windows 3.1 or 3.11 which have OpenNET network access (available via the PCLINK2 networking adapter).
  - Between iRMX systems and PCs running Windows 3.1 or 3.11 on top of the iRMX for Windows OS.
  - Between iRMX and Windows 3.1 or 3.11 running simultaneously within the same system using the iRMX for Windows OS.

    Both the PCLINK2 network adapter and the iRMX for Windows OS provide DOS and Windows with OpenNET network access using a NetBIOS interface. As a result, we refer to this version of the DDE Router as the NetBIOS DDE Router.
- RTE (*routerte.exe* and *rtedde.lib*) - You can use this communications protocol between iRMX and Windows applications running simultaneously within the same system (using the iRMX for Windows OS). Because this facility uses the DOS Real-Time Extension (RTE) facility, you are not required to load networking support on either the iRMX OS or DOS/Windows.

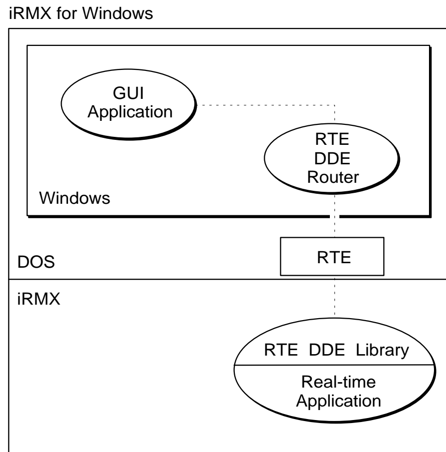



**Figure 7-1. RTE Communication between iRMX and Windows within the Same System**

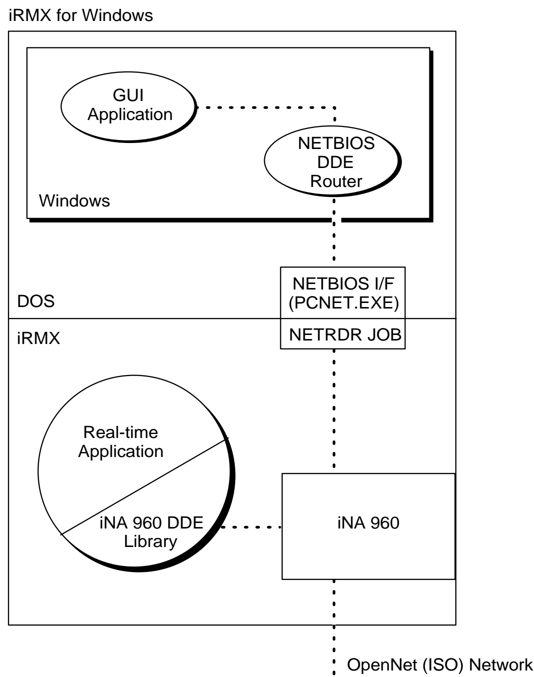



**Figure 7-2. NetBIOS Communication between iRMX and Windows within the Same System**

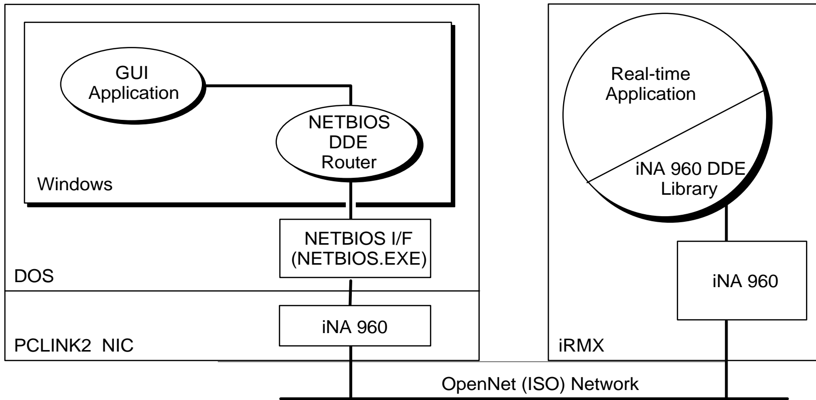



**Figure 7-3. NetBIOS Communication between iRMX and Windows on Different Systems**

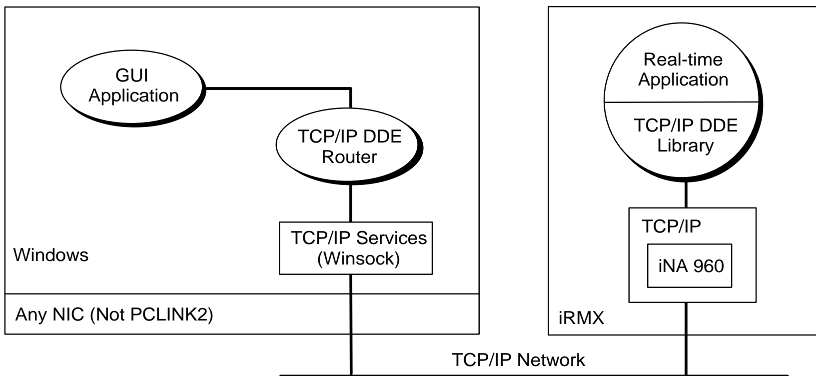



**Figure 7-4. TCP/IP Communication between iRMX and Windows on Different Systems**

## iRMX DDE Capabilities

A single DDE-aware iRMX application is capable of the following:

- It can only communicate with Windows-based DDE clients and servers. It cannot communicate with other DDE-aware iRMX applications.
- It can only use a single communications protocol to communicate with Windows-based DDE clients and servers. For example, if an iRMX application is bound to the DDE Library supporting TCP/IP-based communications, it cannot also be bound to the library supporting OpenNET communications.
- As a DDE client, it can communicate with any number of Windows-based servers running on any number of Windows-based systems. Further, it can have any number of conversations established with each of these servers.
- As a DDE server, it can offer only a single service to Windows-based clients. Therefore, it can invoke the **server_dde_register** function only once.
- As a DDE server, it can accept and support communications with any number of Windows-based clients. Further, it may support any number of conversations with each of these clients.

Windows-based DDE applications do not have the above restrictions. Each application can be implemented independent of the communications protocol and can use more than one protocol. Additionally, Windows-based DDE servers can offer any number of services and can support any number of clients accessing each of these services.

## iRMX DDE Restrictions

There are certain restrictions to be aware of when using the iRMX DDE facility:

- You can only use the iC-386 C compiler to create DDE-aware iRMX applications. You cannot use third-party compilers. In addition, each DDE-aware iRMX application must bind to its own copy of the appropriate DDE Library. Therefore, two or more applications cannot share the same copy of a library.

  DDE is a low-performance communications protocol. This is not due to the underlying networking protocols or its usage with iRMX. It is a result of the way messages are passed within Windows. The messages passed between Windows processes can contain only two data items, a long (32-bit) word and a short (16-bit) word. This is not sufficient to pass all of the information necessary to a DDE transaction.

To pass this information, which includes application, topic, item and data strings, the sending process must establish global atoms, which provide unique references for the strings. Since atoms are themselves referenced using name strings, establishing and referencing them require additional system overhead.

- The iRMX DDE implementation supports only the CF_TEXT data type. There is no support for any type of binary data.

### DDE Router Internal Configuration Limits

The internal configuration limits of the DDE Router are:

- Only the CF_TEXT ASCIIZ string data type is supported.
- The size of CF_TEXT ASCIIZ strings (including the terminating null) cannot exceed 512 bytes in length.
- The size of the command string in the **client_dde_execute** function (including the terminating null) cannot exceed 512 bytes in length.
- Data item names, application names and topic names (including the terminating null) cannot exceed 32 bytes in length.

## Configuring the Windows DDE Routers

The standard Windows configuration file, *win.ini*, contains the configuration parameters for the DDE Routers in the [DDERouter] section.

A DDE Router needs these three pieces of information in order to support a DDE conversation: a network name, a separator character, and an application name.

A DDE Router must recognize that a request to establish a conversation (a DDE_INITIATE message) is for a remote server. Client applications indicate this by filling the application name string with the network name for the system on which the application is executing as well as the name associated with that application.

The DDE Router recognizes that both pieces of information are present by searching for the special character that separates them in the string. By default, this separator character is "%". If you use a different character, for example the "|" character, add a statement, such as "Separator=|" to the [DDE Router] section of the *win.ini* file.

When you use the OpenNET protocol, the NetBIOS DDE Router must establish a unique name for itself on the network. This name must not conflict with any names in use on the network. To specify this name, add a statement of the form "PCName=*name*" in the *win.ini* file. If you do not specify a name, the default name is "WINDOWS".

⟹ **Note**

If you use the TCP/IP protocol, the TCP/IP DDE Router does not require this statement and ignores it if present. In all cases, the TCP/IP protocol uses the name established for the machine during the installation/configuration of the TCP/IP facility. Similarly, the RTE protocol does not require a name because communication is between entities within the same system

When you use the RTE protocol in an isolated iRMX for Windows system, the RTE DDE Router must know which network name indicates that the target (server) is an iRMX application running within the same system. To specify this name, add the statement "RMXName=*name*" to the *win.ini* file. If you do not specify a name, the default name is "RFW".

## Enabling DDE Routing on the Windows System

You can start a DDE Router automatically when you start Windows or manually from within Windows.

To start a DDE Router automatically after invoking Windows, add the following line to the [Windows] section of the *win.ini* file.

```
load=router_name
```

where *router_name* is either *routerte.exe* (RTE), *routetcp.exe* (TCP/IP), or *routenb.exe* (OpenNET).

To start a DDE Router manually from within Windows:

1. If necessary, create a new Windows group. In the Program Manager, go to the File menu. Select NEW, select the PROGRAM ITEM button, select the PROGRAM GROUP button and then click the OK button. Enter "iRMX DDE" as the Description Name and then click the OK button again.
2. Now create an icon for the DDE Router. In the Program Manager window, go to the File menu and select "New".
3. Click the Program Item button and then click the OK button.

4. In the Description box, enter a description for the desired router. For example, "TCP/IP DDE Router". In the Command Line box, enter the pathname for the desired router. On iRMX for Windows systems, these files are located in the *\dosrmx* directory. For Windows-only systems, you must copy the executables from your iRMX system.
5. The icon for the DDE Router appears. Double-click on this icon to start DDE Router.

You can also start more than one DDE Router. To do this when loading a DDE Router automatically, you can add multiple **load** statements. If you are loading a DDE Router manually, repeat the steps to get a startup icon for each DDE Router you will use.

> **Note**
> You can also start DDE Routers automatically when Windows initializes by placing icons for the DDE Routers in the "Startup" Program Group.

## Preparing the Windows Environment for NetDDE

To use the OpenNET protocol, you must start the appropriate NetBIOS service before starting Windows.

- For standalone Windows machines which utilize the PCLINK2 Network Interface Card (NIC), you must load the networking software onto the NIC and start up the NetBIOS service. Use the command:

  ```
  net start rdr pc_name
  ```

  where *pc_name* specifies a unique network name for the DOS machine.

- For machines running iRMX for Windows, you must initialize the iRMX OS, iRMX networking, the DOS networking redirector, and the iRMX NetBIOS interface for DOS before starting up Windows.

  1. In file *:config:loadinfo*, uncomment the **sysload** commands for the iNA 960 networking job required by your networking adapter or service, the iRMX-NET jobs, and the *netrdr* job.

     See also: Network Jobs, *System Configuration and Administration*

2. After starting iRMX, initialize the DOS NetBIOS interface using the command at the DOS prompt:

   **net start netrdr *pc_name***

   where *pc_name* specifies a unique network name for the DOS machine. *pc_name* must not be the same name assigned to iRMX-NET server or client in the */net/data* file.

To use the TCP/IP protocol, you must perform certain steps to install and configure the TCP/IP service:

- For Windows 95 and Windows NT, the Windows installation process installs and configures the TCP/IP service.
- For Windows for Workgroups, the TCP/IP service is a separate package called "TCP/IP-32 For Windows For Workgroups." You can obtain this package from an on-line service, such as Microsoft's Internet server. After obtaining this package, follow the package's instructions for installing and configuring.
- For Windows 3.1, the procedure for installing the TCP/IP package you have chosen is described in the included documentation. The package you choose must provide a version of *winsock.dll* compliant to Winsock Specification 1.1.

To use the RTE protocol with iRMX for Windows, you must ensure that the *dosrmx* directory is included in the DOS PATH statement. You must also ensure the RTE DDE Router running under Windows before you start any DDE-aware iRMX applications.

## Preparing the iRMX Environment for NetDDE

In order to use NetDDE from iRMX, you must properly initialize the appropriate networking support.

- For the OpenNET protocol, load and configure the appropriate iNA 960 network job required by your network adapter. You must uncomment the associated **sysload** command in the *:config:loadinfo* file.

  See also: *i*.job, System Configuration and Administration*

- For the TCP/IP protocol, load and configure the TCP/IP kernel. You must also sysload the special DDE TCP/IP support job, *tcpdde.job*.

  See also: *TCP/IP and NFS for the iRMX Operating System,*
  *i*.job, System Configuration and Administration*

- For the RTE protocol, no special preparation is required. You must run the RTE DDE Router before you start any DDE-aware iRMX applications.

# DDE Programming

The following sections describe the different types of interaction that can take place between the iRMX OS and Windows. These interactions fall into two categories: iRMX client to Windows server or Windows client to iRMX server.

## DDE Messages

All DDE conversations are conducted by passing certain defined DDE messages between server and client applications. Since the DDE Router is the interface between the iRMX OS and Windows, these messages are passed between the Router and Windows.

See also: iRMX Client to Windows server Conversations,
Windows Client to iRMX server Conversations, in this chapter

| DDE Message | Description |
|---|---|
| WM_DDE_ACK | Sent in response to a received message. Provides a positive or negative acknowledgment of the message receipt. |
| WM_DDE_ADVISE | Requests the server application to supply a notice for a data item whenever it changes, such as during hot or warm links. |
| WM_DDE_DATA | Sends a data-item value to the client application. |
| WM_DDE_EXECUTE | Sends a string to the server application, which parses it into a series of commands. |
| WM_DDE_INITIATE | Initiates a conversation between the client and server applications. |
| WM_DDE_POKE | Sends a data-item value to the server application. |
| WM_DDE_REQUEST | Requests the server application to provide the data-item value. |
| WM_DDE_TERMINATE | Terminates a conversation. |
| WM_DDE_UNADVISE | Terminates a permanent data link. |

## iRMX Client to Windows Server Conversations

Figure 7-5 illustrates an iRMX client application communicating with a Windows server application using DDE messages. As far as the server application is concerned, the client is just another local Windows application.

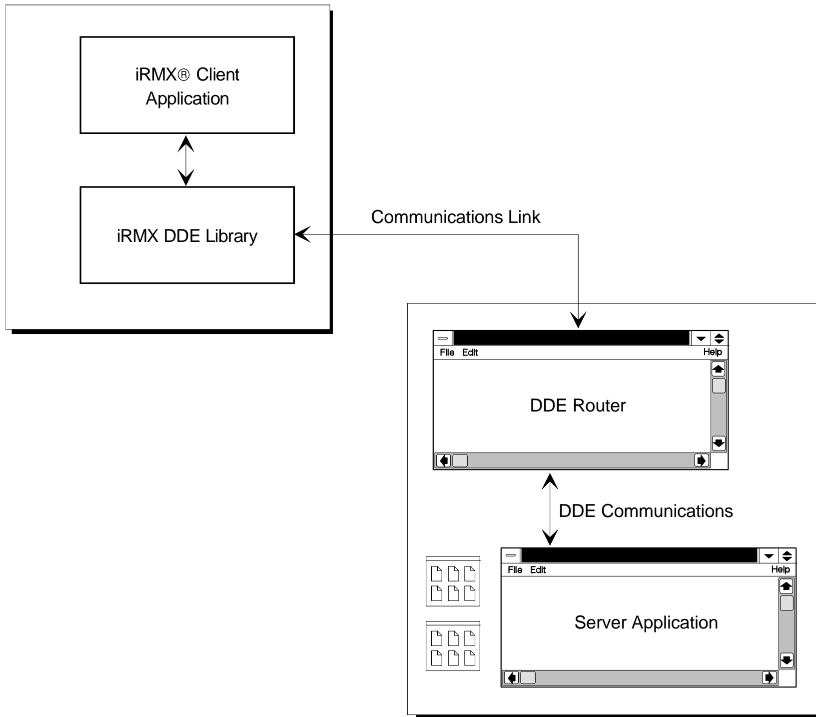



**Figure 7-5. iRMX to Windows DDE Conversations**

An iRMX client application uses the functions listed in Table 7-1.

**Table 7-1.  DDE Library Client Functions**

| Functions | Action |
|---|---|
| dde_library_init | Initialize the local DDE library |
| client_dde_initiate | Request the start of a DDE conversation |
| client_dde_request | Ask server to provide data |
| client_dde_poke | Ask server to accept unsolicited data |
| client_dde_execute | Send a command string to server |
| client_dde_run_application | Start a Windows application |
| client_dde_terminate | Halt a conversation |
| client_link_callback | Accept linked data |
| client_dde_open_hot_link | Ask server to update data whenever it changes |
| client_dde_close_link | Tell server that a data item should no longer be updated |
| client_dde_open_warm_link | Ask server to notify client whenever data changes |

➡ **Note**

Only the DDE-specific parameters of the iRMX DDE library functions are described in these figures.  The *rmxdde.h* header file and the DDE Library section of this chapter completely describe the syntax of all the iRMX DDE library functions.

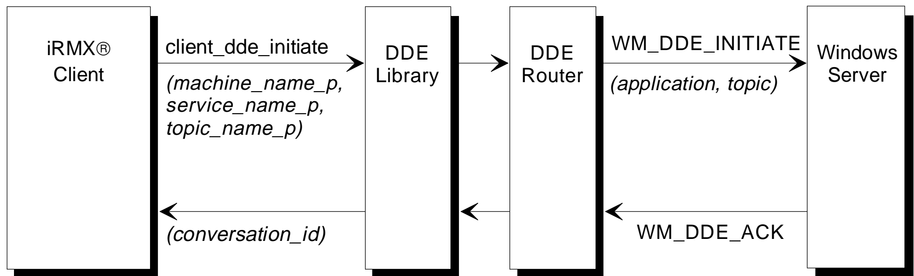
## Initiating Conversations

This section describes how a DDE client application running in an iRMX OS establishes a conversation with a Windows server application.

Before any conversation can be initiated by an iRMX client, the client must ensure that the DDE library has been initialized. If not, the client can initialize it with the **dde_library_init** function.

See also: **dde_library_init** function

The iRMX client invokes the **client_dde_initiate** function to initiate the conversation. The iRMX DDE library responds to this function by sending a message to the appropriate router. The DDE Router broadcasts the request as a **WM_DDE_INITIATE** message if the *win.ini* file on its system contains the same machine name. If the server application on the specified machine acknowledges the initiate request, a `conversation_id` is returned to the client. Other functions use this `conversation_id` to identify this specific conversation between client and server.

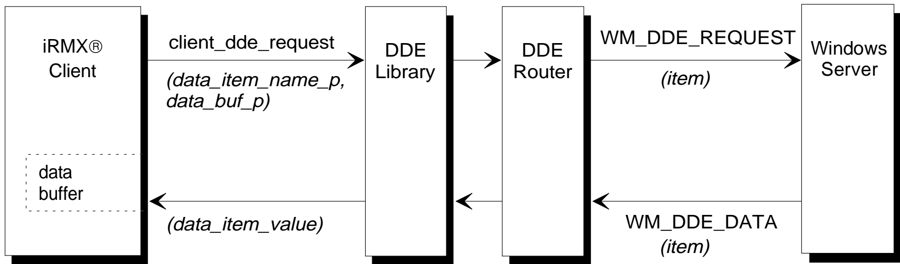


## Transferring Single Items

Once a client has established a conversation with a server, the client can request data items from the server or submit data items to the server.

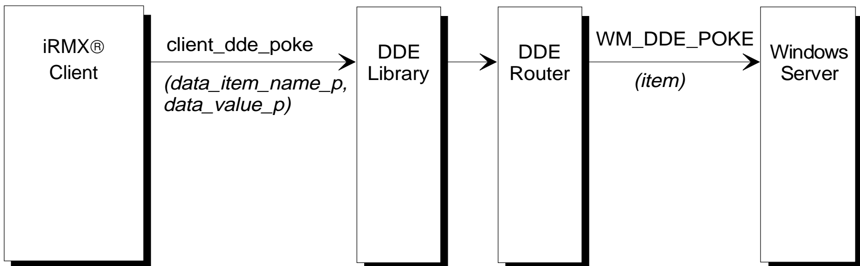
### Requesting Data Items from the Server

To request data items, the iRMX client invokes the **client_dde_request** function. The DDE library passes this request on to the appropriate the DDE Router. The router then acts for the remote iRMX client and sends a **WM_DDE_REQUEST** message to the specified server. After the server application supplies the requested item, the DDE library puts its contents into the data buffer specified by the client in the **client_dde_request** function.



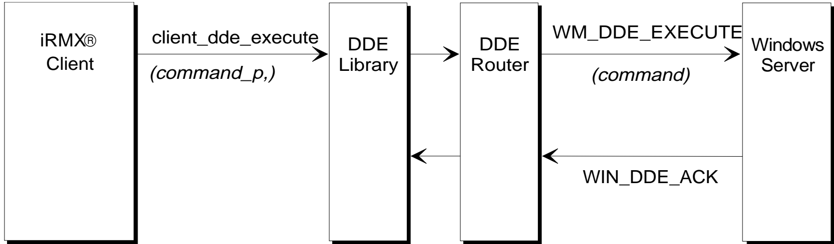

### Submitting Data Items to the Server

To submit data items to the server, the iRMX client invokes the **client_dde_poke** function. The DDE library passes this request on to the appropriate DDE Router. The router then acts for the remote iRMX client and sends a **WM_DDE_POKE** message to the specified server.



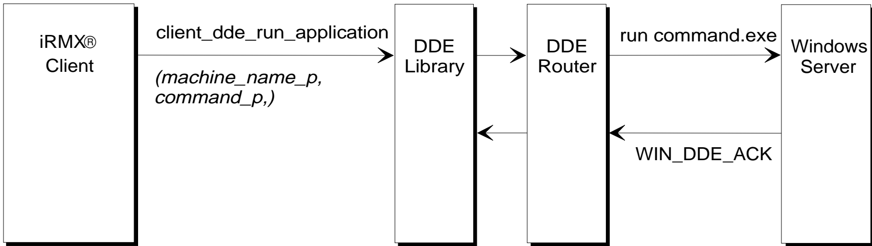
## Submitting Commands to the Server

To submit a string of commands to the server, the iRMX client invokes the **client_dde_execute** function. The DDE library passes this request on to the appropriate DDE Router. The router then acts for the remote iRMX client and sends a **WM_DDE_EXECUTE** message to the specified server.



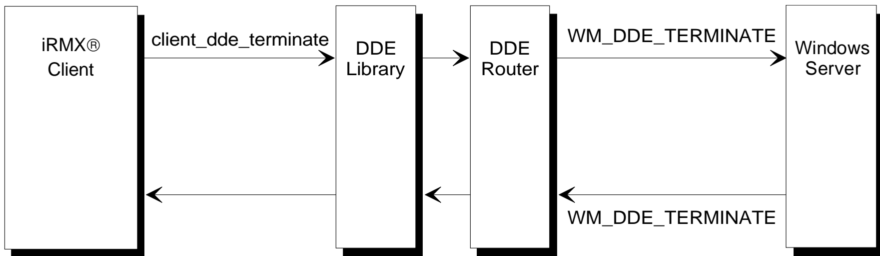

### Running a DDE Windows Application

To run a Windows application, the iRMX client invokes the **client_dde_run_application** function. The DDE library passes this request on to the appropriate DDE Router. The router then starts the Windows application.



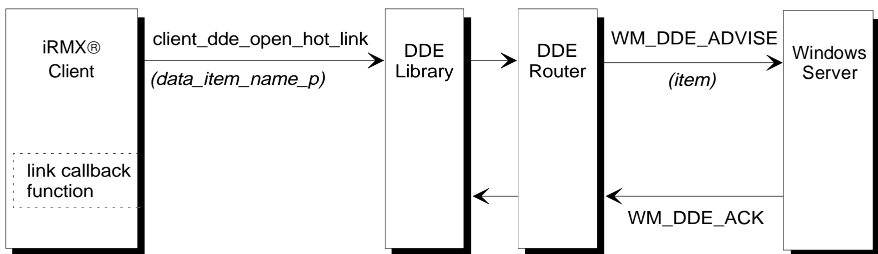
### Terminating a Conversation

To terminate a conversation, the iRMX client invokes the **client_dde_terminate** function. The DDE Router, acting for the client, sends a **WM_DDE_TERMINATE** message to the appropriate server. The server responds with a **WM_DDE_TERMINATE** message of its own and the conversation is ended.

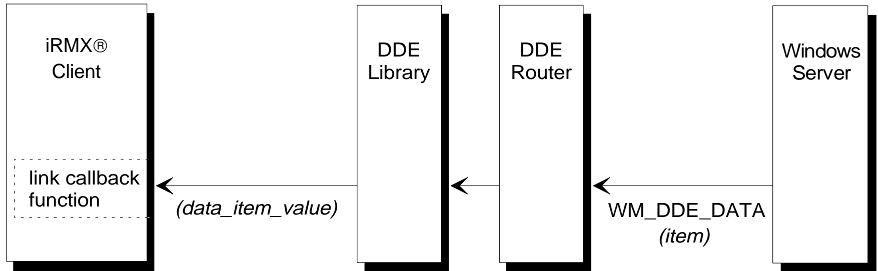



## Using Hot Links

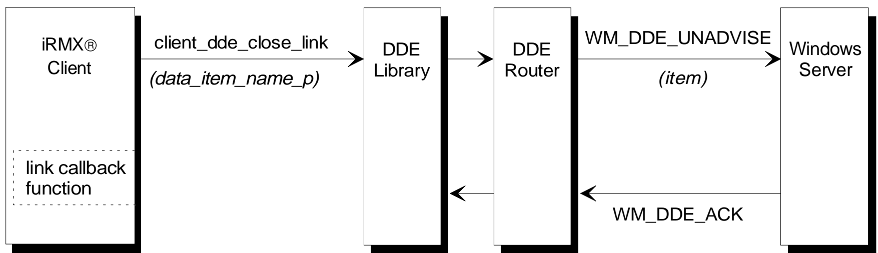
After calling **client_dde_initiate**, (making available a link callback function to the DDE library), an iRMX client can establish a hot link. To do so, the iRMX client invokes the **client_dde_open_hot_link** function. The DDE library passes this request to the DDE Router. The router then acts for the remote iRMX client and sends a **WM_DDE_ADVISE** message to the specified server. The server responds by sending a **WM_DDE_ACK** message indicating that it has access to the requested data item.



Acting for the Windows server, the DDE Router passes any **WM_DDE_DATA** messages on to the DDE library. The DDE library, having been given the pointer to the client's link callback function when the client invoked **client_dde_initiate**, now invokes that callback function.



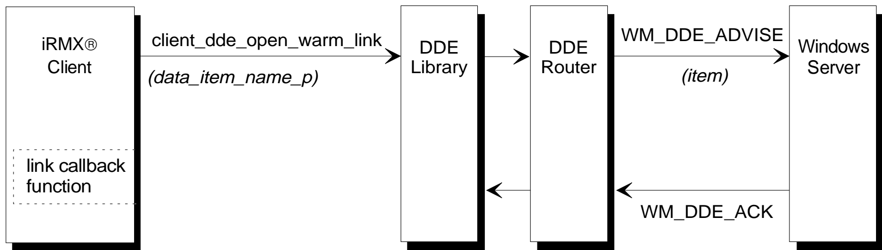

When the client no longer needs updates to the specified data item, it invokes the **client_dde_close_link** function. The DDE library passes this request on to the DDE Router. The router then sends a **WM_DDE_UNADVISE** message to the server.



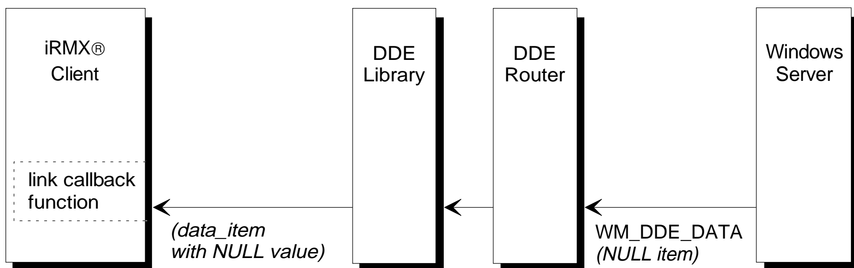

To initiate messages that a server would respond to in a DDE hot link transaction, the iRMX client uses the functions listed in Table 7-1.

## Using Warm Links

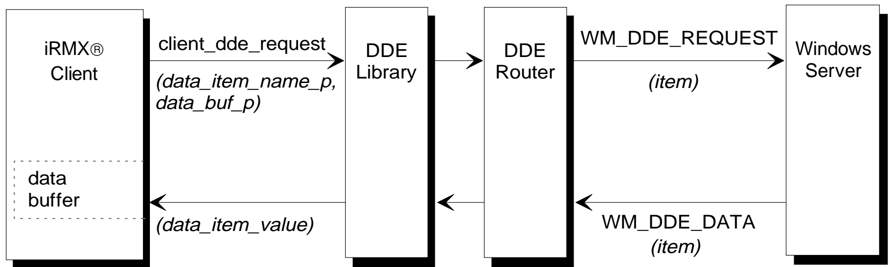
After calling **client_dde_initiate**, (making available a link callback function to the DDE library), an iRMX client can establish a warm link by invoking the **client_dde_open_warm_link** function. The DDE library passes this request to the DDE Router. The router then acts for the remote iRMX client and sends a **WM_DDE_ADVISE** message to the specified server. The router passes a flag with the message indicating that a warm link is requested rather than a hot link.



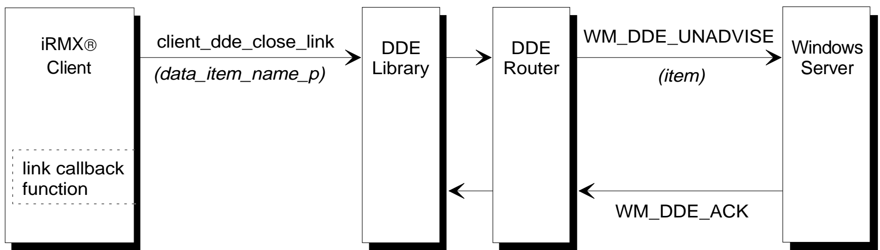

Similar to a hot link conversation, the DDE Router and the DDE library process any **WM_DDE_DATA** messages, and the client's link callback function is invoked. The callback function signals the iRMX client application, which must then issue a **client_dde_request**.



The client invokes a **client_dde_request** function whenever it wants the actual data item.  The callback function cannot invoke the **client_dde_request** function.




To end the warm link, the iRMX client invokes the **client_dde_close_link** function as it does in a hot link conversation.



## Windows Client to iRMX Server Conversations

Figure 7-6 illustrates a Windows client application communicating with an iRMX server application using DDE messages. As far as the client application is concerned, the server is just another local Windows application.

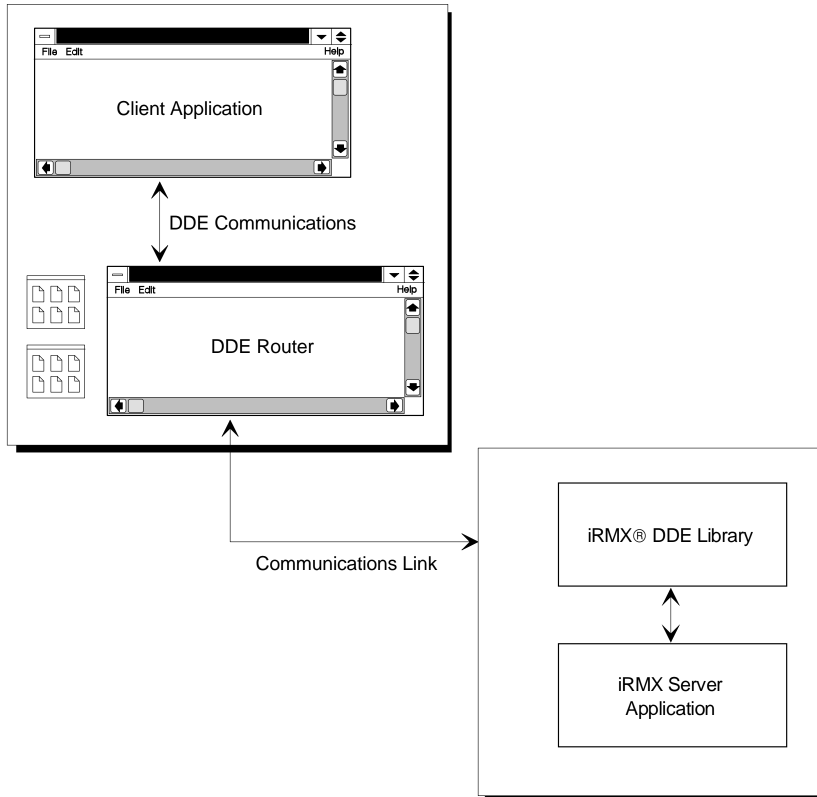



**Figure 7-6. Windows to iRMX Conversations**

An iRMX server uses the functions listed in Table 7-2.

**Table 7-2. DDE Library Server Functions**

| Functions | Action |
|---|---|
| dde_library_init | Initialize the local DDE library |
| server_dde_register | Notify the DDE library that the application is a server |
| server_dde_update_link | Notifies the DDE library that a linked data item is ready to be sent |
| server_data_callback | Respond to a client's poke, request, or execute message |
| server_conversation_callback | Respond to client's request to start or halt a conversation |
| server_dde_terminate | Halt a conversation that the client cannot halt |

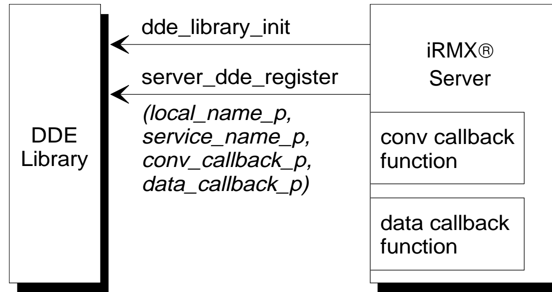
## Establishing Conversations

This section describes how a DDE server application running on an iRMX OS responds to a Windows client application that wants to establish a conversation.

Before any conversation can be accepted by an iRMX server, the server must initialize the DDE library using the **dde_library_init** function and register itself with the library using the **server_dde_register** function.

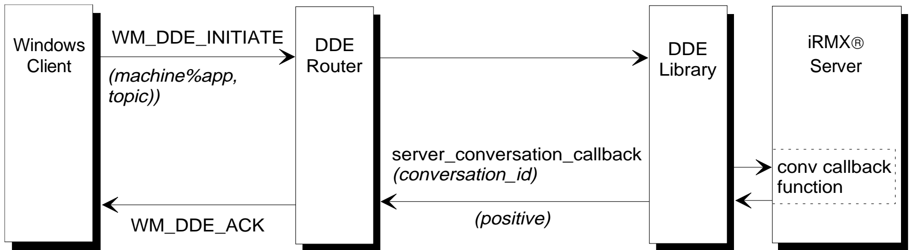
In the registration process, the server posts this information with the library:

- Local machine name
- Service name of server application
- Server conversation callback function
- Server data callback function



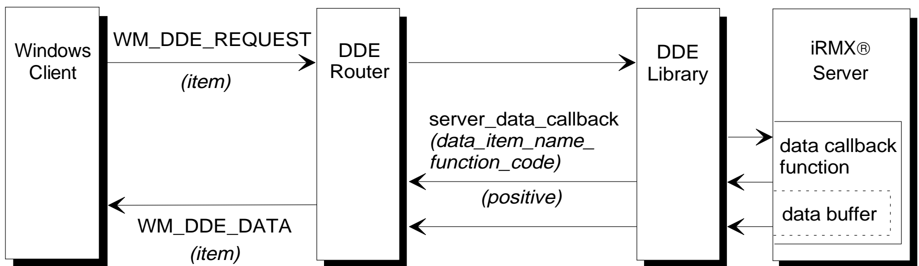

The library invokes the callback functions when appropriate. These sections describe the appropriate conditions.

In response to a **WM_DDE_INITIATE** message, the DDE Router sends a message to the appropriate iRMX machine. The DDE library associated with the requested server invokes the server's conversation callback function. The conversation callback function then accepts or rejects the initiate request. For a positive response, the DDE Router returns a positive **WM_DDE_ACK** message to the client application. At this point a conversation has been established.



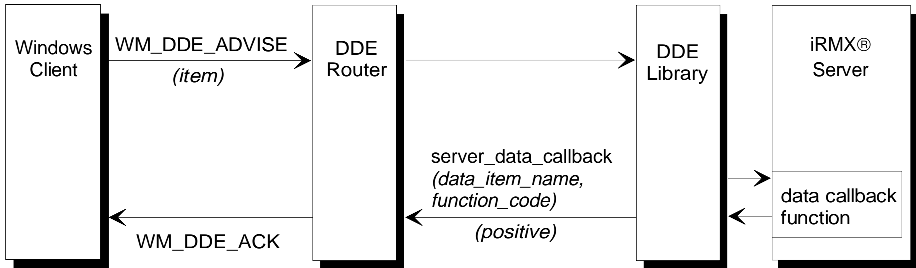

## Responding to Data Item Requests

In response to a **WM_DDE_REQUEST** message, the DDE Router forwards the request to the appropriate DDE library. The library then invokes the **server_data_callback** function that the server had previously posted. The server's callback function is responsible for acknowledging the request and copying the requested data item to a data buffer. Receiving a positive response, the DDE Router takes the information in the data buffer and creates a **WM_DDE_DATA** message to return to the client application.



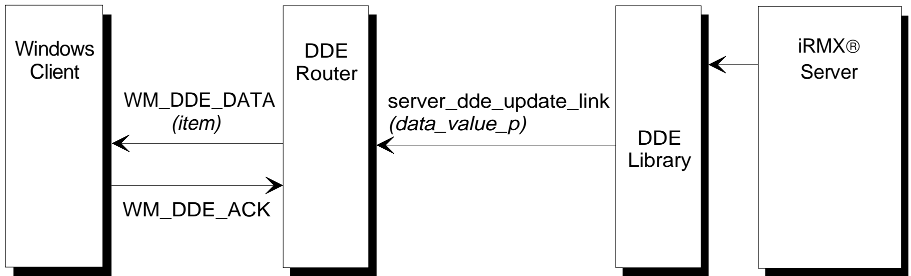
## Responding to Hot Links

In response to a **WM_DDE_ADVISE** message, the DDE Router forwards the request to the appropriate DDE library. The library then invokes the **server_data_callback** function that the server had previously posted. The library uses the `function_code` parameter to tell the server that it needs to respond to a DDE hot link. The callback function is responsible for acknowledging the request. Receiving a positive response, the DDE Router returns a **WM_DDE_ACK** message to the client application.

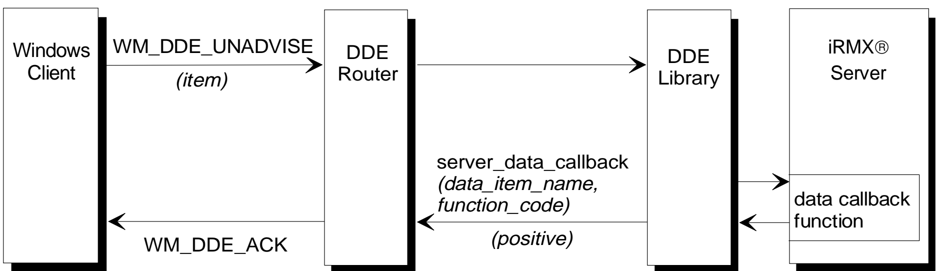



If the iRMX server responds negatively to a **WM_DDE_ADVISE** message, the DDE Router returns a **WM_DDE_ACK** negative message to the client application.

Whenever the value of the hot-linked data item changes, the iRMX server can invoke the **server_dde_update_link** function to pass notification and data back to the client using the DDE library. The DDE library passes the `data_value_p` information on to the DDE Router. In turn, the DDE Router sends a **WM_DDE_DATA** message to the client application. If the client responds with a **WM_DDE_ACK** message, the DDE Router has no response.

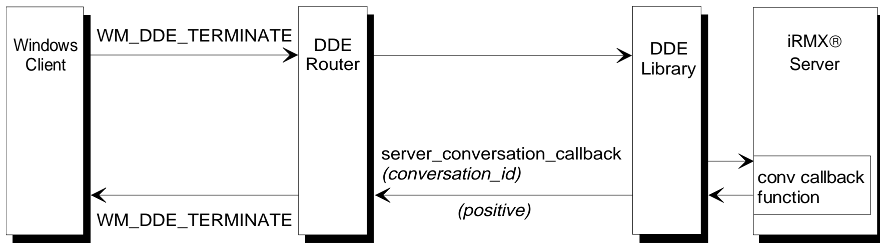



In response to a **WM_DDE_UNADVISE** message, the DDE Router forwards the request to the appropriate DDE library. The library then invokes the **server_data_callback** function for the server, using the `function_code` parameter to indicate that a DDE close link action is requested. The DDE Router returns a **WM_DDE_ACK** message for the client after it receives a positive acknowledgment from the server's data callback function.



## Handling Termination Requests

In response to a **WM_DDE_TERMINATE** message, the DDE Router forwards the request to the appropriate DDE library. The library then invokes the conversation callback function for the server identified by the specified `conversation_id`. The DDE Router generates a **WM_DDE_TERMINATE** message for the client after it receives a positive acknowledgment from the conversation callback function. If for some reason the server suspects that the client cannot send a **WM_DDE_TERMINATE** message due to some error, the server can invoke the **server_dde_terminate** function to terminate the conversation.



# DDE Library

This section contains reference information about the iRMX DDE Library.

## Summary of DDE Library Functions

Any iRMX applications that want to act as a DDE server or client must use the DDE library functions described in this section. Table 7-3 lists the functions by operation type. The functions are described in alphabetical order.

**Table 7-3. DDE Library Summary**

| Operations | Function | Description |
|---|---|---|
| Mandatory | dde_library_init | Initializes the iRMX DDE Library |
| Simple Client | client_dde_initiate | Initiates a DDE conversation with a DDE server application on a specified machine |
| | client_dde_terminate | Terminates an established DDE conversation |
| | client_dde_poke | Pokes a data item value in a DDE server application |
| | client_dde_request | Obtains a data item value from the DDE server |
| | client_dde_execute | Submits a string of commands to a DDE server |
| Client Link | client_dde_open_hot_link | Requests the DDE server to send a specified data item value whenever it changes |
| | client_dde_open_warm_link | Requests the DDE server to send notification whenever a specified data item value changes |
| | client_dde_close_link | Terminates the link (hot or warm) to the DDE server |
| | client_link_callback | Response to a server_dde_update_link function |
| Misc. Client | client_dde_run_application | Starts a Windows program |
| Server | server_dde_register | Notifies the DDE library that the application invoking this function is a DDE server |
| | server_dde_update_link | Notifies the DDE library that a linked data item is ready to be sent |
| | server_dde_terminate | Terminates a DDE conversation established with it by a specified client |
| | server_conversation_ callback | Response to a DDE client's attempts to initiate or terminate a conversation |
| | server_data_callback | Response to an attempt to poke, request, or execute data or attempts to initiate or close a link |

➠ **Note**

This example bind order applies only to the iRMX *make* utility:

```
bnd386                             & BND386 bind invocation
:sd:intel/lib/cstart32.obj,        & C startup code
(objects),                         & application code
:sd:rmx386/lib/router.lib,         & DDE libraries
:sd:intel/lib/cifc32.lib,          & shared C library
:sd:rmx386/lib/udiifc32.lib,       & udi interface library
:sd:rmx386/lib/rmxifc32.lib        & system call interface
(bind controls)                    & library
```

Where `router.lib` is:

The *tcpdde.lib* library if you are using TCP/IP. You must also include the *:sd:intel/lib/net3c.lib* and *:sd:intel/lib/socket3c.lib* support libraries.

The *inadde.lib* library if you are using OpenNET.

The *rtedde.lib* library if you are using RTE.

## Error Codes

Many of the DDE library functions take an argument which is a pointer to a status word set by the DDE library. The status word may return any standard iRMX error code or a special error status returned by the library. Error codes returned by the library are encoded as shown in Table 7-4.

**Table 7-4. Error Codes**

| Error | Value | Meaning |
|---|---|---|
| E_OK | 0H | No error occurred |
| E_EXIST | 6H | A link parameter is wrong. For example, a machine name in client_dde_initiate does not answer, or the machine name in server_dde_register is already in use. |
| E_TRANSMISSION | 0BH | A message cannot be delivered to the network. For example, a hardware error occurred or Windows is not listening, etc. |
| DDE_NOT_OK | 0FFFFH | A table is full or a bad parameter was passed. |
| (other codes) | | iRMX exception codes may also be returned. For example, a DDE library call might internally use a semaphore, and if that fails, the resulting condition code is returned as status.<br><br>See also: Condition codes, *System Call Reference* |

## DDE Library Functions

These pages contain descriptions of all the DDE library functions. The DDE library functions rely upon the DDE header file, *rmxdde.h*, located in the *intel**include*\ directory.

# client_dde_close_link

Cancels the request for asynchronous notification of changes in the value of a data item established on either a warm link or a hot link.

See also: **client_dde_open_hot_link** function
**client_dde_open_warm_link** function

## Syntax

```
void client_dde_close_link (WORD conversation_id,
       char *data_item_name_p, WORD *status_p);
```

## Parameters

`conversation_id`
Identifies the conversation that involves the data item that is no longer needed. The conversation was established by the **client_dde_initiate** function.

`data_item_name_p`
Points to a string that contains the data item name.

`status_p`
Points to a value that contains the DDE status.

# client_dde_execute

Requests that a string of commands be submitted to a DDE server for execution. This is a synchronous function.

## Syntax

```
void client_dde_execute (WORD conversation_id, char *command_p,
       WORD *status_p);
```

## Parameters

`conversation_id`
: Identifies the DDE server that will receive the string of commands. The conversation was established by the **client_dde_initiate** function.

`command_p`
: Points to a string that contains the commands to be executed by the DDE server.

`status_p`
: Points to a value that contains the DDE status.

# client_dde_initiate

Initiates a DDE conversation on a specific topic with a DDE server application on a specified machine. This is a synchronous operation.

## Syntax

```
conversation_id = client_dde_initiate (char *machine_name_p,
        char *service_name_p, char *topic_name_p,
        LINKFUNCPTR link_callback_p, WORD *status_p);
```

## Return Value

`conversation_id`

A unique conversation identifier for one instance of a conversation on a particular machine. This identifier is used in subsequent calls.

## Parameters

`machine_name_p`

Points to a string specifying the name for the remote machine. When using the NetBIOS DDE Transport, the name must match that assigned to the Windows machine using the `pcname` parameter (in *win.ini*). When using the TCP/IP DDE Transport, the name must be the Internet name assigned to the machine you want to talk to. Names are case-insensitive when using either the NetBIOS or TCP/IP DDE Transports. When using the RTE DDE Transport, the name is not important. In this case, the DDE library can only communicate with the RTE DDE Router running under Windows on the same machine.

`service_name_p`

Points to a string containing the service name of the DDE server application.

`topic_name_p`

Points to a string containing the topic name for the DDE conversation.

`link_callback_p`

Points to a client's link notification callback function. This may be null if no links will be established.

`status_p`

Points to a value that contains the DDE status.

## Additional Information

The client application may establish multiple conversations with the same DDE server application.

See also: **client_dde_open_hot_link** function and
**client_dde_open_warm_link** function

# client_dde_open_hot_link

This function requests the DDE server to send the updated value of a specified data item whenever it changes.

## Syntax

```
void client_dde_open_hot_link (WORD conversation_id,
       char *data_item_name_p, WORD *status_p);
```

## Parameters

`conversation_id`

A word that identifies the conversation whose topic contains the requested data item. The conversation was established by the **client_dde_initiate** function.

`data_item_name_p`

Points to a string that contains a data item name. The client wants the server to return the value of this data item each time the value changes.

`status_p`

Points to a value that contains the DDE status.

## Additional Information

The client application is notified of changes in the specified data item using a callback function that it provided to the DDE library with the **client_dde_initiate** function. The library invokes the callback function and provides the function with the new value of the data item that it received from the server.

If the DDE server is an iRMX application, it uses the **server_dde_link** function to send the new value to the library.

# client_dde_open_warm_link

This function requests the DDE server to send notification whenever the value of a specified data item changes.

## Syntax

```
void client_dde_open_warm_link (WORD conversation_id,
       char *data_item_name_p, WORD *status_p);
```

## Parameters

`conversation_id`
: A word that identifies the conversation whose topic contains the requested data item. The conversation was established by the **client_dde_initiate** function.

`data_item_name_p`
: Points to a string that contains a data item name. The client wants notification from the server each time the value of this data item changes.

`status_p`
: Points to a value that contains the DDE status.

## Additional Information

The client application is notified of changes in the specified data item using a callback function that it provided to the DDE library with the **client_dde_initiate** function. The library invokes the callback function but it does not give the function the new value of the data item. Instead, the client must issue a **client_dde_request** function if it requires the new value of the data item. The **client_dde_request** function call may not be made in the callback function.

If the DDE server is an iRMX application, it uses the **server_dde_link** function to notify the library when the value changes.

# client_dde_poke

This function pokes a data item value in a DDE server application. This is a synchronous operation.

See also: **client_dde_initiate** function

## Syntax

```
void client_dde_poke (WORD conversation_id,
       char *data_item_name_p, char *data_value_p,
       WORD *status_p);
```

## Parameters

`conversation_id`
A word that identifies the conversation whose topic contains the data item to be poked. The conversation was established by the **client_dde_initiate** function.

`data_item_name_p`
Points to a string that contains a data item name of the server. The client wants to poke a value into this data item.

`data_value_p`
Points to a string that contains the value to be poked into the specified data item.

`status_p`
Points to a value that contains the DDE status.

# client_dde_request

This function obtains the value of a data item from the DDE server. This is a synchronous operation.

## Syntax

```
void client_dde_request (WORD conversation_id,
       char *data_item_name_p, char *data_buf_p,
       WORD data_buf_size, WORD *status_p);
```

## Parameters

`conversation_id`
A word that identifies the conversation whose topic contains the data item requested. The conversation was established by the **client_dde_initiate** function.

`data_item_name_p`
Points to a string that contains a data item name of the server. The client wants the value of this data item.

`data_buf_p`
A client application buffer area. The DDE library copies the retrieved data item value into this buffer area.

`data_buf_size`
The maximum size of the data item value that may be retrieved. An error is returned if the data item value including the terminating null character does not fit in the buffer.

`status_p`
Points to a value that contains the DDE status.

## Additional Information

An iRMX server uses the **server_data_callback** function to respond to a **client_dde_request** function.

# client_dde_run_application

This function enables an iRMX application to start a Windows program. This is a synchronous operation.

## Syntax

```
void client_dde_run_application (char *machine_name_p,
      char *command_p, WORD flags, WORD *status_p);
```

## Parameters

machine_name_p
: Points to a string specifying the name for the remote machine. When using the NetBIOS DDE Transport, the name must match that assigned to the Windows machine using the pcname parameter (in *win.ini*). When using the TCP/IP DDE Transport, the name must be the Internet name assigned to the machine you want to talk to. Names are case-insensitive when using either the NetBIOS or TCP/IP DDE Transports. When using the RTE DDE Transport, the name is not important. In this case, the DDE library can only communicate with the RTE DDE Router running under Windows on the same machine.

command_p
: Points to a string containing the command line to be executed.

flags
: A word containing one of these: DDE_NORMAL, DDE_MAXIMIZED, DDE_MINIMIZED to determine the initial state of the application's window.

status_p
: Points to a value that contains the DDE status.

# client_dde_terminate

This function terminates a previously established DDE conversation.

## Syntax

```
void client_dde_terminate (WORD conversation_id,
       WORD *status_p);
```

## Parameters

`conversation_id`
A word that identifies the conversation to be terminated. The conversation was established by the **client_dde_initiate** function.

`status_p`
Points to a value that contains the DDE status.

# client_link_callback

The DDE library invokes this function after being notified using a **server_dde_update_link** function that a linked data item is ready to be sent.

## Syntax

```
void client_link_callback (WORD conversation_id,
       char *topic_name_p char *dde_data_item_p,
       char *dde_data_value_p);
```

## Parameters

`conversation_id`
: A word that identifies the conversation whose topic contains the data item to send. The client specified this identifier in a previous **dde_open_warm_link** or **dde_open_hot_link** function.

`topic_name_p`
: Points to a string containing the topic name for the DDE conversation.

`dde_data_item_p`
: Points to a string containing the data item name. The pointer to this string points to a DDE library buffer and must not be retained and used by the application after the callback function returns.

`dde_data_value_p`
: Points to a string containing the data item value. The pointer to this string points to a DDE library buffer and must not be retained and used by the application after the callback function returns. This parameter is undefined in callbacks for warm links.

## Additional Information

The client application makes this function available to the DDE library using the **client_dde_initiate** function.

This function is of type `LINKFUNCPTR`, as defined in *rmxdde.h*.

# dde_library_init

This function initializes the local instance of the iRMX DDE library.

## Syntax

```
void dde_library_init (CONFIGBUF *config_buf_p,
       WORD *status_p);
```

## Parameters

`config_buf_p`
Reserved.

`status_p`
Points to a value that contains the DDE status.

## Additional Information

An iRMX application must invoke this function before an iRMX client application makes any calls or before an iRMX server application makes or responds to any calls. Invoke this function only once.

**Note**
When using the RTE communications protocol, the initialization fails if the RTE DDE Router is not running under Windows.

# server_conversation_callback

The DDE library invokes this function when a DDE client attempts to initiate or terminate a conversation.

## Syntax

```
status = server_conversation_callback (char *client_name_p,
       char *service_name_p, char *topic_name_p,
       WORD conversation_id, WORD function_code);
```

## Return Value

`status`

The application must return a WORD value of 0 if the conversation is accepted or a value of 0XFFFFH if not accepted.

## Parameters

`client_name_p`

Points to a string containing the client machine name. The pointer to this string points into a DDE library buffer and must not be retained and used by the application after the callback function returns.

`service_name_p`

Points to a string containing the requested service name. The pointer to this string points into a DDE library buffer and must not be retained and used by the application after the callback function returns.

`topic_name_p`

Points to a string containing the requested topic name. The pointer to this string points into a DDE library buffer and must not be retained and used by the application after the callback functions returns.

`conversation_id`

A word identifying this DDE conversation. The application must save this for use in subsequent calls.

`function_code`

Set to `DDE_INITIATE` or `DDE_TERMINATE` to specify the requested action.

## Additional Information

The server application makes this function available to the DDE library using the **server_dde_register** function, defined as type `LINKFUNCPTR` in *rmxdde.h*.

# server_data_callback

The DDE library invokes this function when a DDE client attempts to poke, request, or execute data or attempts to initiate or close a link.

## Syntax

```
status =  server_data_callback (char *client_name_p,
       char *service_name_p, char *topic_name_p,
       WORD conversation_id, char *data_item_name_p,
       char *data_buf_p, WORD data_buf_size,
       WORD function_code);
```

## Return Value

`status`

The application must return a WORD value of 0 if the conversation is accepted or a value of 0XFFFFH if not accepted.

## Parameters

`client_name_p`

Points to a string containing the client machine name.  The pointer to this string points to a DDE library buffer and must not be retained and used by the application after the callback function returns.

`service_name_p`

Points to a string containing the requested service name.  The pointer to this string points to a DDE library buffer and must not be retained and used by the application after the callback function returns.

`topic_name_p`

Points to a string containing the requested topic name.  The pointer to this string points into a DDE library buffer and must not be retained and used by the application after the callback function returns.

`conversation_id`

A word identifying this DDE conversation.  The conversation was allocated in an earlier invocation of the server conversation callback.

`data_item_name_p`

Points to a string containing the data item name.

`data_buf_p`

A buffer area that contains the value of the specified data item if the client is attempting to write a data item. If the client is attempting to read a data item, the server must fill in the requested data in this buffer.

`data_buf_size`

The size in bytes of the buffer specified by `data_buffer_p`. This value is only valid and relevant when a client is attempting to read a data item value. For a `DDE_REQUEST`, the string placed into the data buffer must not exceed this value.

`function_code`

Set to `DDE_POKE`, `DDE_REQUEST`, `DDE_EXECUTE`, `DDE_WARM_LINK`, `DDE_HOT_LINK`, or `DDE_CLOSE_LINK` to specify the requested action.

## Additional Information

For convenience, the application is given the client machine name, service name and topic name. All of these are uniquely identified by the conversation identifier and are redundant.

The server application makes this function available to the DDE library using the **server_dde_register** function.

# server_dde_register

This function notifies the DDE library that the application invoking this function is a DDE server.

## Syntax

```
void server_dde_register (char *local_machine_p,
       char *service_name_p, CONVFUNCPTR conv_callback_p,
       DATAFUNCPTR data_callback_p, WORD *status_p);
```

## Parameters

`local_machine_p`
: Points to a string specifying the local machine name.  This parameter is important only when using the NetBIOS DDE Transport, where the library can track a unique machine name for each server.

`service_name_p`
: Points to a string containing the service name of the server application.

`conv_callback_p`
: Points to a server conversation callback function.

`data_callback_p`
: Points to a server data callback function.

`status_p`
: Points to a value that contains the DDE status.

## Additional Information

The server application is notified of requests to open a conversation using a conversation callback function.  The server is notified of requests to read and write data items and to create hot links and warm links using a data callback function.

**Note**
When using the TCP/IP Transport, the registration fails if the special DDE TCP/IP support job (*tcpdde.job*) is not loaded before the registration attempt.

# server_dde_terminate

This function enables an iRMX application acting as a DDE server to terminate a DDE conversation established to it by a specified client. This is a synchronous operation.

## Syntax

```
void server_dde_terminate (WORD conversation_id,
      WORD *status_p);
```

## Parameters

`conversation_id`
: A word that identifies a conversation that the server needs to terminate. The conversation was returned to the server in an earlier invocation of the conversation callback.

`status_p`
: Points to a value that contains the DDE status.

## Additional Information

This function is not necessary in normal operation since clients normally terminate conversations. However, servers can also use this command to terminate conversations if errors occur.

# server_dde_update_link

This function enables an iRMX application that has registered itself as a DDE server and accepted a hot link or a warm link from some DDE client, to notify the DDE library that a linked data item is ready to be sent. This is a synchronous operation.

## Syntax

```
void server_dde_update_link (WORD conversation_id,
       char *data_item_name_p, char *data_value_p,
       WORD *status_p);
```

## Parameters

`conversation_id`
A word that identifies the conversation whose topic contains the data item to be sent. The identifier was returned to the server in an earlier invocation of the conversation callback.

`data_item_name_p`
Points to a string that contains the data item name.

`data_value_p`
Points to a string that contains the value to be sent to the DDE server. This must be null if the link is a warm link.

`status_p`
Points to a value that contains the DDE status.

# Establishing DDE Communications Between Windows and iRMX Applications

This section describes iRMX and Windows example applications that communicate with one another using the DDE protocol. They can run on either the same system or on two separate systems over a network. The iRMX applications are built using the iC-386 compiler and the iRMX DDE Library. The *\rmx386\demo\dde* directory contains the examples and the *make* files for creating their executables. The *readme.txt* file in this directory contains a description of the examples.

## DC Motor Example

The example uses DC motor control to illustrate the use of DDE software. As shown in Figure 7-7, a DC motor is one of the devices participating in Process A that resides on some local or remote PC, Multibus I, or Multibus II system. The object is to provide an operator interface that enables high-level monitoring, and control of the motor through supervisory commands. This example is restricted to the motor although the operator might also be interested in monitoring and controlling other devices.

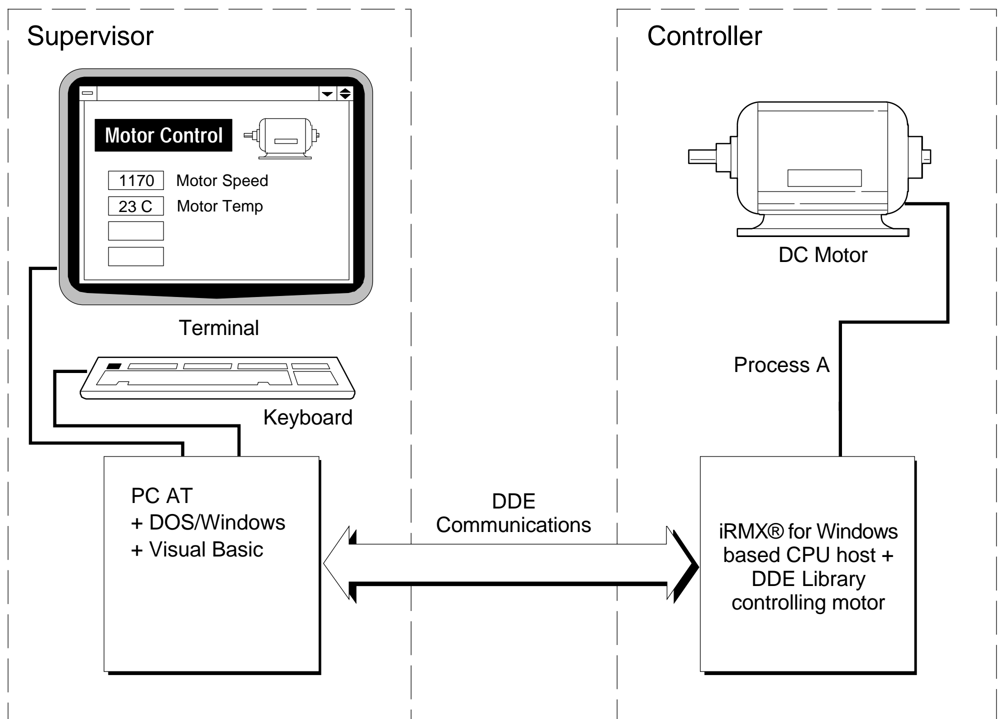



**Figure 7-7. Supervisor/Controller Communications**

These DDE data items are defined for the motor:

- `Powerswitch` - On/off state of the motor. Valid values are `on` and `off`.
- `Desiredspeed` - Desired motor speed.
- `Actualspeed` - Actual speed of the motor. During normal operating conditions, `actualspeed` should equal `desiredspeed`.
- `Motortemp` - Temperature of the motor.
- `Bearingtemp` - Temperature of the bearings.
- `Motorload` - Load on the motor, in amps.
- `Msg` - Textual information indicating alarm or out-of-range conditions, or any other special information to be conveyed to the supervisor.

## DC Motor Description

The supervisor determines the values for `powerswitch` and `desiredspeed`. The values are sent to the controller whenever they change. The controller locally monitors the values of `actualspeed`, `motortemp`, `bearingtemp`, and `motorload`. These values are sent to the supervisor whenever they change (however, how often they are actually reported is an implementation detail, since there could be high overhead in reporting, say, each slight change in `motorspeed`). Lastly, the controller monitors the value of `msg` and reports the value to the supervisor.

Figure 7-8 shows the general appearance of the Visual Basic supervisor window.

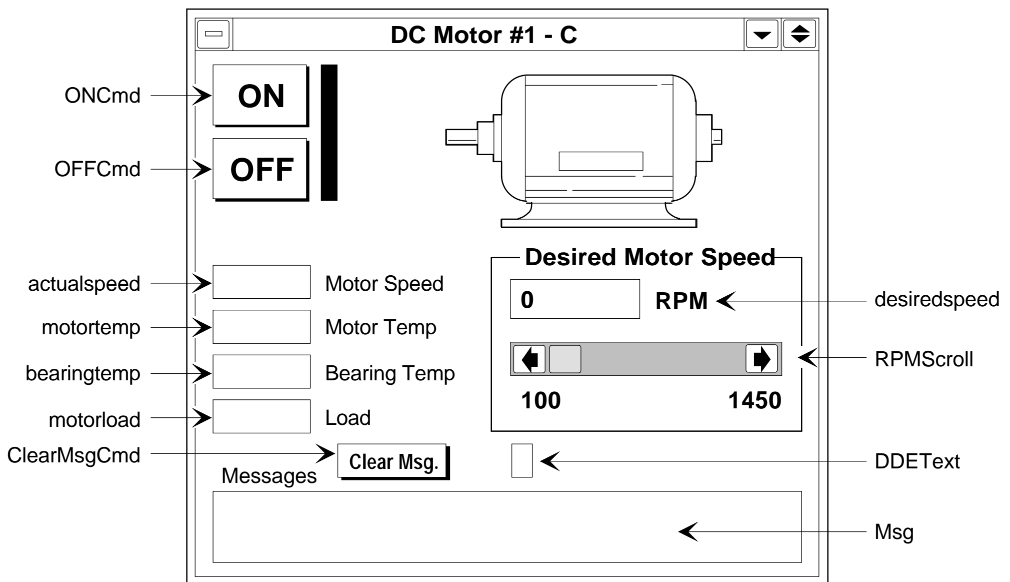



**Figure 7-8. The Visual Basic Supervisor**

## DDE Client Supervisor / DDE Server Controller Implementation

In this example, the iRMX-based controller is implemented as a DDE server and the Visual Basic Windows-based supervisor is implemented as a DDE client. The supervisor initiates all conversations with the controller. All control parameters, such as `powerswitch` state and `desiredspeed`, are poked from the supervisor whenever they change. Status information, such as the actual motor speed, is received by the supervisor through hot links whenever these values change.

Start with this model (iRMX DDE Server) for your own applications, since it affords the most decoupling between your iRMX application and the particular implementation of the Window-based supervisor.

## Visual Basic Supervisor Implemented as DDE Client

The *\rmx386\demo\dde\vb* directory contains the Visual Basic files for the client-supervisor, *cmotor.**. There is no main code. The implementation is entirely event-oriented, with a function or subroutine defined for each control present on the form. The first event that occurs is the loading of the form. Subsequent events occur when controls are activated and messages are received.

**Note**
Only the make file and the source file are provided for the Visual Basic application. The Visual Basic software must be purchased separately.

## iRMX Controller Implemented as a DDE Server

The controller is a loadable iRMX program that simulates direct motor control. In the DDE server model of the controller, the supervisor pokes the `powerswitch` and `desiredspeed` values to the controller. Whenever one of these values changes, a new value is poked and printed to the server screen. To simulate changing status values on the controller, a prompt enables you to enter new values for `actualspeed`, `motortemp`, `bearingtemp`, `motorload`, and `msg`. These new values are then sent as link data to the supervisor, causing the window to be updated.

The *\rmx386\demo\dde* directory contains the C source code and *smotor* generation files for the server-controller.

## Running the Example

The Visual Basic example, *cmotor.exe*, is located in the *\rmx386\demo\dde\vb* directory.

Run the example by performing these steps:

1. On the iRMX system:

   A. Start the iRMX OS.

   B. Start all networking/communications jobs necessary to support DDE transport requirements.

   See also: Preparing the iRMX Environment for NetDDE, in this chapter

   C. To start the iRMX DDE application, enter:

   **smotor *network_name* <CR>**

   where the ***network_name*** parameter, necessary only if using the OpenNET protocol, specifies the name of the server which is to be used across the network.

2. On the Windows system:

   A. If using OpenNET, start the appropriate NetBIOS service.

   See also: Preparing the Windows Environment for NetDDE, in this chapter

   B. Start Windows.

   C. If the appropriate DDE Router has not been started, start it by clicking on its icon.

3. To start the Visual Basic application, enter:

   **cmotor.exe *machine_name* <CR>**

   where *machine_name* depends on the network communications protocol:

| | |
|---|---|
| OpenNET | Same name specified above for the server |
| TCP/IP | Internet name of the machine executing the server |
| RTE | iRMX machine specified in *rmx.ini* using the `rmxname` parameter |

## DDE Server Supervisor/DDE Client Controller Implementation

In this example, the iRMX-based controller is implemented as a DDE client and the Visual Basic Windows-based supervisor is implemented as a DDE server. The controller initiates all conversations with the supervisor. All control parameters, such as the `powerswitch` and `desiredspeed`, are obtained from the supervisor by hot-linking the associated data items with the supervisor so that the controller receives the new values whenever they change. Status information, such as the actual motor speed, is poked to the supervisor whenever the controller needs to inform the supervisor of changes.

### Visual Basic Supervisor Implemented as a DDE Server

The *\rmx386\demo\dde\vb* directory contains the Visual Basic files for the server-supervisor, *smotor.**. There is no main code. The implementation is entirely event-oriented, with a function or subroutine defined for each control present on the form. The first event that occurs is the loading of the form. Subsequent events occur when controls are activated and messages are received.

### iRMX Controller Implemented as a DDE Client

The controller is a loadable iRMX program that simulates direct motor control. In the DDE client model of the controller, `powerswitch` and `desiredspeed` values are obtained by establishing hot links with the supervisor. Whenever one of these values changes, the client callback function is invoked, and the new value is printed to the screen. To simulate changing status values on the controller, a prompt enables you to enter new values for `actualspeed`, `motortemp`, `bearingtemp`, `motorload`, and `msg`. These new values are then poked to the supervisor, causing the window to be updated.

The *\rmx386\demo\dde* directory contains the C source code and *cmotor* generation files for the client-controller.

## Running the Example

Run the example by performing these steps:

1. On the Windows system:

   A. If using OpenNET, start the appropriate NetBIOS service.

   See also: Preparing the Windows Environment for NetDDE in this chapter

2. Start Windows.

3. If the appropriate DDE Router was not automatically started, start it by clicking on its icon.

4. Start the Visual Basic DDE server application, *smotor.exe*.

5. To start the iRMX DDE client enter:

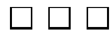
   **cmotor *pc_name* smotor motor <CR>**

   where *pc_name* depends on the network communications protocol:

   | | |
   |---|---|
   | OpenNET | Use the name specified in the Windows system's *pc_name* parameter of the [DDERouter] section of the *win.ini* file. |
   | TCP/IP | Use the Internet name assigned to the Windows system |
   | RTE | Use any name since the field is ignored. |

6. To exit the iRMX DDE client application, enter:

   **exit <CR>**



# iRMX for Windows Default Configuration A

This appendix lists the pre-configured options in the software definition file, used to generate the iRMX for Windows boot image. If you ported an existing application to iRMX for Windows, you may need to alter it to run within the pre-configured software. If your application is incompatible with this configuration, use the Interactive Configuration Utility to change it.

See also: Definition files, *ICU User's Guide and Quick Reference*, for a listing of the definition files you can customize.

Tables of pre-configured options are provided for these system requirements and sub-systems:

- Sub-Systems
- Memory
- Human Interface
- Application Loader
- Extended I/O System
- Basic I/O System
- Device Drivers
- System Debug Monitor
- Nucleus
- Nucleus Communication Service
- VM86 Dispatcher Reserved Interrupts

## Sub-System Configuration

**Table A-1. Sub-Systems Options**

| Sub-Systems | Default |
|---|---|
| Universal Development Interface | Yes |
| Shared C Libraries | No |
| Human Interface | Yes |
| Application Loader | Yes |
| Network Access | No |
| Extended I/O System | Yes |
| Basic I/O System | Yes |
| System Debug Monitor | Yes |
| System Debugger | No |
| OS Extension | Yes |

## Memory Configuration

**Table A-2. Memory Options**

| Memory for System | Default |
|---|---|
| Start Address | 110000H |
| End Address | 1FFFFFH |
| **Memory for Free Space** | **Default** |
| Start Address | 0200000H |
| End Address | 0FFFFFFFFH |

# Human Interface Configuration

**Table A-3. Human Interface Options**

| HI Jobs | Default |
|---|---|
| Jobs Minimum Memory | 0H |
| Jobs Maximum Memory | 0FFFFFFFH |
| Numeric Processor Extension Used | Yes |
| **Prefixes** | **Default** |
| Prefix : | :PROG: |
| Prefix : | :UTILS: |
| Prefix : | :UTIL286: |
| Prefix : | :SYSTEM: |
| Prefix : | :LANG: |
| Prefix : | :$: |
| **HI Logical Names** | **Default** |
| Name = WORK | :SD:WORK |
| Name = UTILS | :SD:UTIL386 |
| Name = UTIL286 | :SD:UTIL286 |
| Name = LANG | :SD:LANG286 |
| Name = RMX | :SD:RMX386 |
| Name = INCLUDE | :SD:INTEL/INCLUDE |

# Application Loader Configuration

**Table A-4. Application Loader Options**

| Application Loader | Default |
|---|---|
| All System Calls | Yes |
| Default Memory Pool Size | 0500H |
| Read Buffer Size | 01000H |

# Extended I/O System Configuration

**Table A-5. EIOS Options**

| EIOS | Default |
|---|---|
| Retries on Physical Attachdevice | 0H |
| Default IO Job Directory Size | 200 |
| **Automatic Boot Device Recognition** | **Default** |
| Default System Device Physical Name | C_RMX |
| **Logical Names** | **Default (Device Name, File Driver, Owner's ID)** |
| Logical Name = BB | BB, PHYSICAL, 0H |
| Logical Name = Stream | STREAM, STREAM, 0H |

# Basic I/O System Configuration

**Table A-6. BIOS Options**

| BIOS | Default |
|---|---|
| Attach Device Task Priority | 129 |
| Timing Facilities Required | Yes |
| Timer Task Priority | 129 |
| Connection Job Delete Priority | 130 |
| Ability to Create Existing Files | Yes |
| System Manager ID | Yes |
| Common Update Timeout | 1000 |
| Terminal Support Code | Yes |
| Control-Sequence Translation | Yes |
| Terminal OSC Controls | Yes |
| Tape Support | No |
| BIOS Pool Minimum | 0800H |
| BIOS Pool Maximum | 0FFFFFH |
| Global Clock | ATRT |
| Global Clock Name* | |

* The Global Clock Name has a blank string as a default.

# Device Drivers Configuration

**Table A-7. Device Drivers Options**

| Driver | Default |
|---|---|
| AT Serial Driver | |
| DUIB Name | COM1 |
| Interrupt Level | 048H |
| Base Port Address | 03F8H |
| Reset Character | 0H |
| Interrupt Character | 0H |
| DUIB Name | COM2 |
| Interrupt Level | 038H |
| Base Port Address | 02F8H |
| Reset Character | 0H |
| Interrupt Character | 0H |
| ROM-BIOS Based Hard Disk Driver | |
| DUIB Name | C_RMX and D_RMX (first iRMX partition)<br>C_RMX0 and D_RMX0 (whole physical drive)<br>C_RMX1 through C RMX4 and D_RMX1 through D_RMX4 |
| Base I/O Port Address | 01F0H |
| Control/Status Port Address | 03F6H |
| ROM-BIOS Based Diskette Driver | |
| DUIB Name | A and B (5.25 inch format, 360 Kbyte)<br>AH and BH (5.25 inch format, 1.2 Mbyte)<br>AM and BM (3.5 inch format, 720 Kbyte)<br>AMH and BMH (3.5 inch, 1.44 Mbyte)<br>AMO and BMO (3.5 inch, 2.88 Mbyte) |
| Interrupt Timeout | 01770H |
| EDOS File Driver | |
| DUIB Name | A_DOS, ... ,Z_DOS |
| DOS File Driver | |
| DUIB Name | C_DOS, ... ,Z_DOS |

# System Debug Monitor Configuration

**Table A-8. System Debug Monitor Options**

| System Debug Monitor | Default |
|---|---|
| Console Port | System Console Primary |

# Nucleus Configuration

**Table A-9. Nucleus Options**

| Nucleus | Default |
|---|---|
| Number of GDT Entries | 8000 |
| Number of IDT Entries | 256 |
| Parameter Validation | Yes |
| Root Object Directory Size | 200 |
| Default Exception Handler | SDB |
| NMI Exception Handler | IGNORE |
| NMI Enable Byte | 4 |
| Exception Handler for Stack Exception | SDB |
| Name of Ex Handler Object Module | |
| Exception Mode | NEVER |
| Low GDT/LDT Slot Excluded from FSM | 0 |
| High GDT/LDT Slot Excluded from FSM | 0 |
| Round Robin Priority Threshold | 140 |
| Round Robin Time Quota | 5 |
| Report Initialization Errors | YES |
| Maximum Data Chain Elements | 0 |
| Nucleus Communication Service | YES |

## Nucleus Communication Service Configuration

**Table A-10. Nucleus Communication Service Options**

| Nucleus Communication Service | Default |
|---|---|
| Message Task Priority | 128 |
| Deletion Task Priority | 128 |
| Default Number of Port Transactions | 16 |
| Default Host ID | 0 |
| Validate Buffer Parameters | Yes |
| Max. No. of Simultaneous Transactions | 080H |
| Max. No. of Simultaneous Messages | 0100H |
| Receive Fragment Failsafe Timeout | 0400H |
| Number of Trace Messages | 255 |

## VM86 Dispatcher Reserved Interrupts Configuration

**Table A-11. DOS Extender Reserved Interrupts**

| DOS Extender Reserved Interrupts | Default |
|---|---|
| Master Level 0 | Clock |
| Master Level 2 | Slave PIC |

# Index


## A

Above Board, Intel, 12
admin.exe file, 54
ASCIIZ string type, 60
automatic DDE startup, 66

## C

callback function
conversation callback, 82, 85, 101
data callback, 82
link callback, 77, 91, 99
registering, 104
cmotor.exe file, 111
cold link, 60
cold link conversation
initiating, 72
responding to, 81
config, loadinfo file, 55
config.nia file, 54
conversations, 60
CPU registers, 27

## D

data, structure, DOS, 43
data buffer, 103
data transfer, 23
DDE
client functions, 71
compiler selection, 64
data buffer, 73, 103
data item name, 102
error codes, 88
library, 81
library summary, 86
message routers, 59
restrictions, 64
server functions, 80
terminology, 60
DDE conversations, 101
cold link, 72, 81
iRMX to Windows applications, 70
warm link, 77
Windows to iRMX applications, 79
DDE data buffer, 82
size, 96, 103
DDE flag
DDE_MAXIMIZED, 97
DDE_MINIMIZED, 97
DDE_NORMAL, 97
DDE function code
DDE_CLOSE_LINK, 103
DDE_EXECUTE, 103
DDE_HOT_LINK, 103
DDE_INITIATE, 101
DDE_POKE, 103
DDE_REQUEST, 103
DDE_TERMINATE, 101
DDE_WARM_LINK, 103
DDE library, buffer, 101
copying retrieved data, 96
initializing, 100
invoking conversation callback function, 101
invoking data callback function, 102
invoking link callback function, 99
notifying, 106
DDE library functions
client_dde_close_link function, 76, 78, 89
client_dde_execute function, 74, 90
client_dde_initiate function, 72, 91
client_dde_open_hot_link function, 75, 93
client_dde_open_warm_link function, 77, 94
client_dde_poke function, 73, 95
client_dde_request function, 73, 96

client_dde_run_application
function, 74, 97
client_dde_terminate function, 75, 98
client_link_callback function, 99
dde_library_init function, 81, 100
server_conversation_callback
function, 101
server_data_callback function, 82, 102
server_dde_register function, 81, 104
server_dde_terminate function, 85, 105
server_dde_update_link function, 106
DDE message
WM_DDE_ACK, 75, 82, 83, 84
WM_DDE_ADVISE, 75, 77, 83
WM_DDE_DATA, 76, 82
WM_DDE_EXECUTE, 74
WM_DDE_INITIATE, 82
WM_DDE_POKE, 73
WM_DDE_REQUEST, 82
WM_DDE_TERMINATE, 85
WM_DDE_UNADVISE, 76, 84
DDE router
automatic startup, 66
manual startup, 66
DDE Windows application, running, 97
default configuration, 115
deleting, DOS programs, 40
deleting objects, 25
deletion handler, 40
extension, 39
demo\c\vm86ext\ directory, 41, 46
device driver, iRMX OS using DOS, 43
dispatcher job restrictions, 40
dlc.exe file, 51, 56
DOS
and iRMX file access, 18
data structure, 43
deleting programs, 40
encapsulated file driver, 18
encapsulated task, 14
interrupt handling, 39
interrupts masked, 50
iRMX OS making requests, 43
iRMX OS using device driver, 43
real-time extensions, 15
DOS RTE jobs, 25
dosdevs.c file, 46
dosext.asm file, 41
dosext.c file, 41
dosrmx directory, 51
dosrtec.lib file, 26
dosrtel.lib file, 26
dosrtes.lib file, 26
dynamic data exchange, *See* DDE
data type, 65
network names, 65
router configuration, 65

## E

EDOS file driver, 18
error codes, 88
examples
DDE server, 107
DOS RTE, 28
installing an extension, 41
making rqe_dos_request call, 47
setting DOS data structure, 46
extensions
deactivating, 39
deletion handler, 39, 40
entry point, 38
restrictions, 40
writing, 37

## F

function code, RTE, 27
function_code, 103

## H

himem.job job, 14
hot link, 60
hot link conversation
closing, 89
initiating, 75
notifying DDE library, 106
opening, 93

## I


i*.job, 21
i*.job file, 68
IDT, 38
iNA 960 network job, 68
inadde.lib file, 61, 87
initializing the DDE library, 100
installing a VM86 extension, 38
intel\include\ directory, 88
interrupt, levels, 37
Interrupt Descriptor Table, 38
interrupt handler
    software, 37
    VM86 Mode-generated, 39
interrupts
    DOS, description of handling, 38
    masked, DOS, 50
    response time, 49
    RTE, 27
    vectored to PVAM handler, 39


## L


loadinfo file, 22, 67
loadname command, 53


## M


machine name, 81, 101
manual DDE router startup, 66
message routers, 59
MS-DOS, 11


## N


net3c.lib file, 87
netrdr.job job, 18, 53
NULL data_value_p parameter, 106


## O


objects, deleting, 25
OpenNET protocol, 61


## P


partitions, DOS and iRMX OS, 18
PC-DOS, 11
pcnet.exe file, 18, 53
PL/M 286 convention, 27
pointer
    parameter, 25
    real mode, 25
pre-configured options, 115
    application loader, 117
    BIOS, 118
    device drivers, 119
    EIOS, 118
    human interface, 117
    memory, 116
    Nucleus, 120
    Nucleus Communication Service, 121
    sub-systems, 116
    System Debug Monitor, 120
    VM86 Dispatcher reserved interrupts, 121
protected mode, 14, 37
PVAM, 37
    interrupt handler, 39


## R


real mode
    pointer, 25
    stack, 27
real-time extensions, 15
registering iRMX DDE server, 104
restrictions, extension, 40
rls.job job, 51, 53, 55
rmx.ini file, 22
rmx386\demo\dde\vb directory, 111
rmx386\demo\rte\lib\ directory, 25
rmx386\demo\rte\obj\ directory, 28
rmx386\demo\wterm directory, 36
rmx386\jobs\ directory, 14
rmx4win.dll file, 25
rmxc.h file, 26
rmxdde.h file, 71, 99, 101
rmxext.c file, 41
rmxfuncs.obj file, 25
rmxintfc.h file, 26
rmxtsr.exe file, 14

ROM BIOS, iRMX OS making requests, 43
routenb.exe file, 61
router.exe file, 67
routetcp.exe file, 61
rqe_dos_request call, 40, 43
rqe_read_segment call, 23
rqe_set_vm86_extension call, 37
rqe_write_segment call, 23
rqegetrmxstatus call, 25
RTE, 15
    calls, 25
    function code, 27
    functions, 28
    invoking, 27
    relation to Nucleus system calls, 25
    restrictions, 27
    software interrupt request, 27
rtedde.lib file, 87

## S

separator character, 65
service information, inside back cover
service name, 81, 101, 102
setname command, 53
smotor.* file, 112
smw.job job, 14
socket3c file, 87
stack, real mode, 27
string types, 60
system calls
    BIOS restrictions, 40
    Nucleus restrictions, 40

## T

tcp/ip protocol, 61
tcpdde.job file, 104
tcpdde.lib file, 61, 87
topic name, 60, 101, 102
transferring data, 23

## V

Virtual 86 Mode, 14
Visual Basic DDE application, 113
VM86 dispatcher, 37
VM86 extension, installing, 38

## W

warm link, 60
warm link conversation
    closing, 89
    notifying the DDE library, 106
    opening, 94
win.ini file, 72
winsock.dll file, 68
wlc.exe file, 51, 56
writing extensions, 37
wterm file, 36

iRMX® Programming Concepts
for DOS and Windows
469154-003

# *WE'D LIKE YOUR OPINION*

| Please rate the following: | Excellent | Good | Fair | Poor |
| --- | --- | --- | --- | --- |
| ■ Manual organization | ❒ | ❒ | ❒ | ❒ |
| ■ Technical accuracy | ❒ | ❒ | ❒ | ❒ |
| ■ Completeness | ❒ | ❒ | ❒ | ❒ |
| ■ Clarity of concepts and wording | ❒ | ❒ | ❒ | ❒ |
| ■ Quality of examples and illustrations | ❒ | ❒ | ❒ | ❒ |
| ■ Overall ease of use | ❒ | ❒ | ❒ | ❒ |

Comments: ____________________________________________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

Please list any errors you found (include page number): ____________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

____________________________________________________________

Name ____________________________________________________________

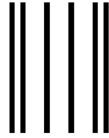
Company Name ____________________________________________________

Address _________________________________________________________

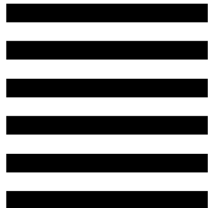
____________________________________________________________

May we contact you? ____________________ Phone ____________________________

Thank you for taking the time to fill out this form.



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

**OPD Technical Publications, HF2-72**
**Intel Corporation**
**5200 NE Elam Young Parkway**
**Hillsboro, OR 97124-9978**

Please fold here and close the card with tape. Do not staple.

# WE'D LIKE YOUR COMMENTS....

This document is one of a series describing Intel products. Your comments on the other side of this form will help us produce better manuals. Each reply will be reviewed. All comments and suggestions become the property of Intel Corporation.

If you are in the United States and are sending only this card, postage is prepaid.

If you are sending additional material or if you are outside the United States, please insert this card and any enclosures in an envelope. Send the envelope to the above address, adding "United States of America" if you are outside the United States.

Thanks for your comments.

# International Sales Offices

**AUSTRALIA**
Intel Australia Pty. Ltd.
Unit 1A
2 Aquatic Drive
Frenchs Forest, NSW, 2086
Sydney

Intel Australia Pty. Ltd.
711 High Street
1st Floor
East Kw. Vic., 3102
Melbourne

**BRAZIL**
Intel Semiconductores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
CEP 01311-Sao Paulo - S.P.

**CANADA**
Intel Semiconductor of Canada, Ltd.
999 Canada Place
Suite 404, #11
Vancouver V6C 3E2
British Columbia

Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Ontario

Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Ontario

Intel Semiconductor of Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Quebec

**CHINA/HONG KONG**
Intel PRC Corporation
China World Tower, Room 517-518
1 Jian Guo Men Wai Avenue
Beijing, 100004
Republic of China

Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway
Central
Hong Kong

**FINLAND**
Intel Finland OY
Ruosilantie 2
00390 Helsinki

**FRANCE**
Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex

**GERMANY**
Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany

**INDIA**
Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001

**ISRAEL**
Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

**ITALY**
Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano

**JAPAN**
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

Intel Japan K.K.
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192

Intel Japan K.K.
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360

Intel Japan K.K.
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222

Intel Japan K.K.
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 460

**KOREA**
Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010

**MEXICO**
Intel Technologica de Mexico
S.A. de C.V.
Av. Mexico No. 2798-9B, S.H.
44620 Guadalajara, Jal.,

**NETHERLANDS**
Intel Semiconductor B.V.
Postbus 84130
3009 CC Rotterdam

**RUSSIA**
Intel Technologies, Inc.
Kremenchugskaya 6/7
121357 Moscow

**SINGAPORE**
Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130

**SPAIN**
Intel Iberia S.A.
Zurbaran, 28
28010 Madrid

**SWEDEN**
Intel Sweden A.B.
Dalvagen 24
171 36 Solna

**TAIWAN**
Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei

**UNITED KINGDOM**
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ

# If you need to contact Intel Customer Support

Contacting us is easy. Be sure that you have the following information available:

- Your phone and FAX numbers ready
- Complete description of your hardware or software configuration(s)
- Your product's product code
- Current version of all software you use
- Complete problem description

| Type of Service | | How to contact us |
|---|---|---|
| **FaxBACK*** <br> fax-on-demand system <br><br> 24 hrs a day, 7 days a week | Using any touch-tone phone, have technical documents sent to your fax machine. Know your fax number before calling. | U.S. and Canada: (800) 628-2283 <br> (916) 356-3105 <br><br> Europe: +44-1793-496646 |
| **Intel PC and LAN Enhancement Support BBS** <br><br> 24 hrs a day, 7 days a week | Information on products, documentation, software drivers, firmware upgrades, tools, presentations, troubleshooting. | U.S and Canada: (503) 264-7999 <br> Europe: +44-1793-432955 <br> Autobaud detect <br> 8 data bits, no parity, 1 stop |
| **CompuServe*** <br> **Information Service** <br><br> 24 hrs a day, 7 days a week | Worldwide customer support: information and technical support for designers, engineers, and users of 32-bit iRMX® OS and Multibus product families. | Worldwide Locations: <br> (check your local listing) <br><br> Type: GO INTELC once online. |
| **Customer Support** | Intel Multibus Support engineers offering technical advice and troubleshooting information on the latest Multibus products. | U.S. and Canada: (800) 257-5404 <br> (503) 696-5025 <br> FAX: (503) 681-8497 <br> Hrs: M-F; 8-5 PST <br> Europe: +44-1793-641469 <br> FAX: +44-1793-496385 <br> Hrs: M-F; 9-5:30 GMT |
| **Hardware Repair** | Multibus board and system repair. | U.S. and Canada: (800) 628-8686 <br> (602) 554-4904 <br> FAX: (602) 554-6653 <br> Hrs: M-F; 7-5 PST <br> Europe: +44-1793-403520 <br> FAX: +44-1793-496156 <br> Hrs: M-F; 9-5:30 GMT |
| **Sales** | Intel Sales engineers offering information on the latest iRMX and Multibus products and their availability. | Worldwide: Contact your local Intel office or distributor <br> U.S. and Canada: (800) 438-4769 <br> (503) 696-5025 <br> FAX: (503) 681-8497 <br> Hrs: M-F; 8-5 PST |
| **Correspondence** <br> Mail letters to: | Worldwide: <br><br> Intel Customer Support <br> Mailstop HF3-55 <br> 5200 NE Elam Young Parkway <br> Hillsboro, Oregon 97124-6497 | Europe: <br><br> European Application Support <br> Intel Corporation (UK) Ltd. <br> Pipers Way <br> Swindon, Wiltshire <br> England SN3 1RJ |

* Third-party trademarks are the property of their respective owners.